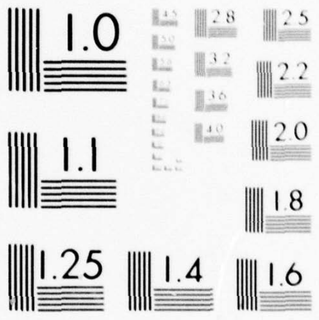
AD-A070 960 GENERAL ELECTRIC CO SUNNYVALE CALIF F/G 9/2
A SIMULATION MODELING APPROACH TO UNDERSTANDING THE SOFTWARE DE--ETC(U)
JUN 79 J A MCCALL, G Y WONG, A H STONE F49620-78-C-0054
UNCLASSIFIED AFOSR-TR-79-0776 NL

1 OF 2
AD
A070960


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

GENERAL ELECTRIC

INFORMATION SYSTEMS PROGRAMS
SUNNYVALE, CALIFORNIA 94086

FICATION

AD A070960

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) AFOSR-TR- 79-0776	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) A SIMULATION MODELING APPROACH TO UNDERSTANDING THE SOFTWARE DEVELOPMENT PROCESS.		5. TYPE OF REPORT & PERIOD COVERED (9) Final rept.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) J.A. McCall, G.Y. Wong, A.H. Stone		8. CONTRACT OR GRANT NUMBER(s) (15) AFOSR F49620-78-C-0054
9. PERFORMING ORGANIZATION NAME AND ADDRESS Information Systems Program General Electric Co Sunnyvale, CA 94086		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) 61102F 2304 A2 (17) A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, D.C. 20332		12. REPORT DATE (11) June 1979
		13. NUMBER OF PAGES 96
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 100p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Development Process, Computer System Simulation, Software Management. Cost Estimation.

149 450 [illegible]

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report, prepared for the Air Force Office of Scientific Research (AFOSR), describes an assessment of the feasibility of utilizing simulation techniques to aid in the management of large-scale software developments. A model of the software development process was constructed, state-of-the-art prototype simulation tools used, and an experiment conducted to demonstrate the feasibility. A result of this effort is the concept of a Software Development Process Simulator which could be utilized to assist in project planning (cost estimation) and project control (progress status assessment).

DD FORM 1 JAN 73 1473

UNCLASSIFIED

14

A SIMULATION MODELING APPROACH TO UNDERSTANDING THE SOFTWARE DEVELOPMENT PROCESS

J. A. McCall
A. H. Stone
G. Y. Wong

June 1979

Prepared for:

Director, Mathematical and Information Sciences
Air Force Office of Scientific Research
ATTN: NM
Building 410, Bolling AFB
Washington, DC 20332

Prepared by:

General Electric Company
~~Information Systems Programs~~
450 Persian Drive
Sunnyvale, California 94086

149 450

FOREWORD

This document is the final technical report for the Software Development Process Simulation Study, contract number F49620-78-C-0054. The contract was performed in support of the Air Force Office of Scientific Research, (AFOSR), Directorate of Mathematical and Information Science.

The report was written by G. Wong, A. Stone, and J. McCall of the Sunnyvale Operations, Information Systems Programs, General Electric Company. The program manager for the study was G. Walters. S. Amaral prepared the final documentation.

Technical guidance was provided by Lt. Col. G. McKemie, AFOSR program manager. Technical discussions were also held with R. Weber and A. Sukert, RADC, and Maj. N. McQuage and CPT. J. Duquette, ESD.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

SECTION 1

EXECUTIVE SUMMARY

1.1 TASK OBJECTIVES

Significant progress has been made during the last few years in identifying the problems and complexities involved with the development of software systems and providing techniques to overcome these obstacles. Several major conferences and workshops in recent years highlight the work that has been done in the research community ([NATO 69], [PROC 73], [FIND 75], [PROC 75]; see Bibliography, Section 7). What has evolved is a more disciplined environment for the production of software. Formal specification, design, and implementation methodologies are being developed. More milestones and visible software products during the development phase have been identified. Software support tools have become more sophisticated in providing assistance in the design and development of software. Considerable error and cost data have been collected and a better understanding of the software development environment is evolving. Cost, productivity, and reliability studies add to this understanding and provide data for prediction and estimation. The factors in software quality and associated metrics are being studied to obtain more quantitative measurements of the quality of a software product. Demonstration projects are being undertaken to prove the effectiveness of new techniques.

All of these R&D efforts contribute to a more disciplined and structured development process. This discipline and structure lends itself to more effective management. Most of the tools and techniques that have resulted from these R&D efforts support micro-level activities within the software development process. Few assist in the management of the entire process. The goal of this proposed effort is to address this void.

A potential management tool, made possible by the more disciplined approaches taken to software development, is a simulation model of the development process. Simulation models traditionally have been used by management for analyses such as system design studies, tradeoff analyses, performance assessments, and impact analyses. A model of the software development process would facilitate these same types of analyses of the development effort itself. The analyses supported by such a tool would span both management planning (cost estimation) and control (progress and impact assessment).

The initial step toward developing a simulation tool to aid in the management of a software development involves developing the concept of such a tool and assessing the feasibility of using simulation techniques to construct a model of the software development process. This report describes the results of this initial step. Specifically, under contract number F-49620-78-C-0054, sponsored by the Air Force Office of Scientific Research, the objectives of this study were to:

1) Determine the feasibility of applying simulation techniques to modelling the software development process.
2) Describe the software development process in a manner conducive to developing a simulation model.
3) Provide insights into modelling specific aspects of the software development process.
4) Discuss the potential benefits and use of such a model.

1.2 REPORT OVERVIEW

This report describes the research conducted to accomplish the objectives described above. The report is organized as follows:

Section 1 describes the task objectives, provides an outline of the report, and summarizes the findings of the research task.

Section 2 provides a brief introduction to cost estimation techniques, presents current analytic models, and compares the analytic approaches to a simulation approach. This section provides the motivation for assessing the feasibility of simulation techniques.

Section 3 expands upon the simulation approach to modelling the software development process. The orientation of the model is discussed and its anticipated inputs and outputs. Also, in this section our conceptual approaches to modelling the software development process are described including how the level of detail at which to model is determined.

Section 4 contains the model description. the decomposition of the software development process into activities, products, and influencing parameters is presented. Our specific concepts of modelling the activities and the utility of the model are also described.

Section 5 provides a demonstration of some of our concepts. A simplified model of a software development is constructed and simulated using some prototype simulation tools. The simplified model was constructed to represent a past Air Force development and the simulated results are compared to the actual results. The design concepts of the simulation tools are described, as well as the details of the demonstration experiments.

Section 6 provides detailed conclusions of the research study and an identification of follow-on research that should be conducted.

Section 7 provides an extensive list of references used in the study. The references have been organized according to the following categories:

- Cost Estimation
- Software Project Management
- Simulation
- Others

1.3 SUMMARY OF FINDINGS

The basic findings of the study are as follows:

- Simulation techniques can be utilized and provide very beneficial insights into understanding and managing the software development process. The major advantages of a simulation model over analytic models are: (1) it provides better visibility/understanding because it is more detailed and models the actual processes being performed and, (2) it is more flexible, allowing the model to be tailored to the specific development. The simulation modelling approach is still dependent on data collection as analytic techniques have been. One advantage, with respect to data collection, is that the simulation model allows you to test your assumptions about what is going on in the development, whereas analytic models do not provide that degree of visibility into the process and, therefore, rely strictly on past data.
- The level of detail at which the model is developed is critical to its effectiveness. We decided to model at a team-level to avoid the great variances found in individuals' performances. One of the biggest problems in past modelling and cost estimation techniques is the inattention paid to the products produced during developments and their quality. A software development in which little effort is spent during design (and, therefore, has a poor design) will have much more effort spent during implementation and test. These interphase dependencies were identified and could not be accurately modelled by regression techniques in a study for the Electronic Systems Division [GRAV 76]. Our approach to including quality considerations in the model and utilizing these considerations to impact the effort required to perform certain activities is an attempt to model phenomenon. The activity progression and product progression model concepts are presented in Section 3.
- The description of a detailed Software Development Process Model provides immediate benefit to managers, as well as, establishing

a framework for simulations. It is possible to orient the simulation model to support both planning and control functions of management. The simulation approach is more supportive of both phases that the analytic approach because it represents the software development process in more detail. The development process was decomposed into 51 generic activities. Subsets of these activities can be selected and interconnected to represent any specific development effort. The activities described are generally those activities performed by the development organization. Formal contractor reviews are also identified. Extension of the model to include other acquisition office activities ([GORD 78], [COSG 78]) would require simply adding activity models representing these activities.

- The prototype simulation tools which support this process of selecting and interconnecting the activities are described and demonstrated in Section 5. A concept of using path expressions to accomplish this was developed during this research project. The prototype simulation tools identified solve user interface, efficiency, and process modelling problems normally associated with traditional simulations.

The concepts that evolved during the study, and the very good results of the simulation experiment provide the necessary support to assess that simulating the software development process is feasible and considerable worthwhile information can be gained from this approach. In the experiment, a past development was modelled and the simulated results were within 4.98% of the actual results.

Further work is necessary, since this was a feasibility study. Development of the simulator and experimentation have been proposed in an overall research plan. The prototype tools developed during this study provide an excellent starting point for the development of the simulator. However, further work is also necessary on the model itself. Data collection and further analysis are required to develop models of all of the activities identified. Section 6 describes what further research is required.

SECTION 2

INTRODUCTION

2.1 BACKGROUND

The software development process is the subject of increasing research and study. The motivating force behind the widespread interest in software engineering is the high cost of software, the fact that more applications are being computerized, and the greater complexity of the software systems being constructed. There are basically two areas of research of the software development process. On the one hand, there is interest in better ways of developing quality software. This is the goal of the "structured revolution" in software development, which has affected all phases of the software development process -- from analysis to specification, to design, to programming, to test. New software development methodologies have emerged in recent years which emphasize (1) the use of software tools, including automated tools to support software development, and also modularized software functions upon which more complex software functions can be built; (2) the use of techniques which reduce software complexity, including the reduction of control flow complexity through structured programming; and, (3) the use of better techniques for documenting system and software design, including the use of graphical and hierarchical means for presenting control and data flow.

On the other hand, there is interest in better ways of managing software developments so that resource and schedule requirements are estimated more accurately, more efficient project organizations are instituted, and project progress is evaluated more accurately. The software development manager is concerned with project planning and analysis, with respect to software cost and schedule estimation, and he is concerned with project management and control, with respect to project status evaluation and problem identification. The management of a software development project is an extremely difficult job because of the continuous influx of new software development methodologies, the scarcity of historical software cost and schedule information, and the lack of reliable tools to support the planning/analysis and management/control tasks.

The software development process is a people-intensive process. Thus, the research in software development methodologies is concerned with making people more productive at developing software, in terms of increasing the rate of software development and decreasing the occurrence of errors. Similarly, the research in software management methodologies is concerned with making the overall project team more productive and efficient by more accurately anticipating project personnel requirements, and more reliably selecting the best project team organization.

The objective of our research is to formulate a model of the software development process that can be used by the software development researcher, or by the software development manager. The software development researcher will use the model to evaluate the impact that changes in software development methodology will have on personnel productivity. The software development manager will use the model to predict the cost and schedule characteristics of a software development project, and to evaluate the impact that alternative project organization strategies will have on overall project team productivity and efficiency.

2.2 ON MODELS

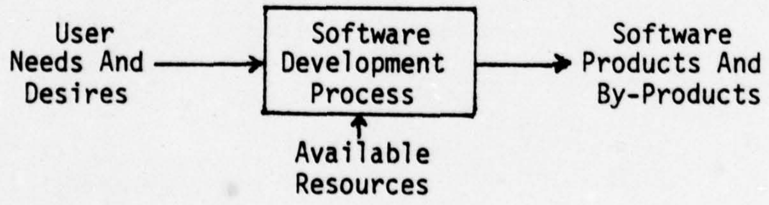
A model is a representation of a system which gathers together in one place our understanding of the behavior of that system. The purpose of developing a model of a system is to have a vehicle for predicting the behavior of the system under various conditions. The adequacy of the model is normally determined by five criteria: (1) applicability -- does the model answer the questions that we want to ask?; (2) confidence

-- is the model sufficiently accurate for our purposes?; (3) completeness -- is the model broad enough to encompass all phenomena of interest?; (4) minimality -- have system states that are unnecessarily discriminated been combined? and; (5) independence -- have system states that involve interacting factors been decomposed into multiple states?

The software development process has been modeled by researchers in software engineering primarily for the purpose of predicting the life cycle costs associated with developing computer software. The models that have been developed are macroscopic models which use analytic techniques to represent the behavior of a software development. We will now consider the merits and shortcomings of these modeling approaches, and explore an alternative approach using simulation techniques. The next section describes the characteristics of the software development process which make it difficult to model, and identifies the analytic modeling approaches that have been utilized in the past. The final section in this chapter discusses how a simulation modeling approach can be applied to modeling software developments. In both sections, the five criteria for determining the adequacy of a model will be applied.

2.3 ANALYTIC MODELING APPROACH

The software development process can be viewed as a black box process which transforms the user's needs and desires, and the available resources into software products and by-products.


User Needs And Desires
Software Development Process
Software Products And By-Products
Available Resources

A macroscopic model of this black box process can be developed by identifying the parameters which affect the progress of the software development, and then constructing analytic relations which can be used to compute the software project cost from the model parameters. The model parameters describe the software system to be developed, that is, the characteristics of the software products and by-products, including system type, size, and complexity. They also describe the characteristics of the resources to be applied in developing the software system, including number of personnel, productivity of personnel, and rate of consumption of computer resources.

The software cost relations describe how the software cost parameters are used to compute the various costs associated with the software development process. These relations must be constructed empirically, using data collected from past software developments. Examples of analytic relations which have been used with some degree of success are the following: (1) <u>similar experience</u> ($C=C_x$) -- where cost data from a previous similar experience are used directly. Similarity of projects is determined by similarity in software cost parameters; (2) <u>statistical</u> ($C= \frac{1}{\omega} \Sigma\omega_i C_i$, where $\omega=\Sigma\omega_i$)-- where the weighted average of cost data from a number of similar experiences is used. The magnitude of each weight is determined by the degree of similarity in software cost parameters; (3) <u>constraint</u> ($C=C_o$) -- where the resources available for the software development are fixed, and the level of effort is adjusted accordingly. This is a design to cost project, where the cost is fixed at C_o; (4) <u>unit of work</u> ($C=\Sigma C_i$) -- where the project is subdivided into sufficiently small pieces such that each piece is equivalent to a single task that an individual can perform over a specified time interval. The cost that is associated with each piece of the project is C_i; (5) <u>quantitative</u> ($C=f(P_1,P_2. . . P_n)$) -- where cost estimating functional relationships are empirically derived. The arguments P_i in these functional relationships are the software cost parameters. For example, average productivity or cost per instruction equations use the estimated system size to generate the predicted project cost.

In order for the software development manager to use a macroscopic model, he must obtain values for the software cost parameters. First, he must determine the characteristics of the software products and by-products. This is in principle very difficult because the target software system does not yet exist! Techniques for accurately estimating the characteristics of the target software system are important because the results of the model are not reliable if the model parameters are inaccurate. Second, he must determine the characteristics of the resources to be applied. This is in principle easy to do because the software development manager can hypothesize a number of resource plans and use the model to assess the feasibility of each plan.

The primary advantage in taking an analytic approach to modeling the software development process is that the resulting macroscopic models are relatively easy to develop and use. To develop an analytic software cost estimating relationship (CER), a set of significant software cost parameters is first identified, and the CER is formulated using analytic techniques such as regression analysis against a historical software cost data base. To use an analytic software CER, values for the significant software cost parameters are estimated and used to evaluate the CER. In addition to convenience, analytic techniques, if automated, are normally highly efficient in terms of processing time.

Despite the convenience of the analytic modeling approach, there are serious shortcomings that limit its usefulness. First, there is limited visibility into the dynamic aspects of the software development process. Analytic models tend to be "black boxes" with only the final project cost and schedule as the model output. Second, analytic models contribute limited understanding of the internal behavior of a software development. Analytic models tend to be "macroscopic" and do not provide insight into the interdependencies between the various software cost parameters. Third, analytic models provide limited decision analysis capabilities. There is little assistance provided for determining how management and development decisions (which cannot be represented quantitatively by a software cost parameter) will affect project cost and schedule.

The adequacy of the analytic approach for modeling the software development process with the objective of supporting the software engineering researcher and the software development manager is summarized in the following table.

Criteria	Evaluation
applicability	limited
confidence	marginal
completeness	limited
minimality	excellent
independence	limited

Applicability is limited because it is difficult to construct analytic models which answer questions pertaining to the detailed progress of a software development. Confidence is marginal because the results of analytic models are not easily extrapolated from one software environment to another. Completeness is limited because only high level macroscopic phenomena are represented in analytic models. Minimality is excellent because of the macroscopic point of view, but independence is very limited, as independent states have been combined together.

2.4 SIMULATION MODELING APPROACH

Whereas the analytic modeling approach treats the software development process as a black box process, the simulation modeling approach attempts to decompose the process and understand the internal behavior of the process. With the simulation modeling approach, we view a software development system as a collection of interdependent elements which act together in a collective effort to achieve the goal of implementing computer software. These elements are primarily people, including analysts, designers, programmers and testers, and machines, including computers and terminals. The simulation view is, thus, a microscopic view, as opposed to the macroscopic view, in the analytic approach. Simulation modeling is the process of developing an internal representation and a set of transformation rules which can be used to predict the behavior of, and relationships between, the set of elements composing the system under study The internal representation of a software development system is described by system state variables, such as software size and complexity, personnel productivity, and project status and progress. The transformation rules describe the interdependence between these system state variables. These transformation rules may be analytic - expressed in the form of functional relationships, or they may be representational - expressed in the form of an algorithm.

Taking the simulation modeling approach basically means having a microscopic view, and being able to express both functional and algorithmic relationships between system state variables. There is a significant increase in descriptive modeling capability when using the simulation approach. In fact, all of the weaknesses associated with the analytic approach -- limited visibility, limited understanding, and limited decision analysis are eliminated with the simulation approach. The modeler can describe the software development process in as much detail as is required to answer the questions that are being investigated.

Unfortunately, although simulation provides a much more powerful tool for studying the software development process, it does not automatically "solve" the problem. There are still a number of difficulties that must be addressed. First, the study of the software development process is still an empirical study. We must still collect data from actual software developments to calibrate and validate our models. Of course, a simulation model will provide insight into the internal behavior of the system, and the interdependencies between system state variables. But, we still have the burden of proof in demonstrating the accuracy of our simulation model. Second, a simulation model of the software development process will, no doubt be parameterized so that it could apply to a number of software development applications in a number of software development environments. Thus, there is the problem of accurately estimating the values of these parameters to be input to the simulation model. This problem existed for the analytic modeler, and it does not go away for the simulation modeler. Third, the amount of effort required to develop a simulation model, and the amount of processing time required to execute a simulation model are substantial. In fact, they are significantly greater than that required for an analytic model.

We will now discuss our basic approach to using simulation modeling for studying the software development process. In Section 4, the details of the development of a simulation model of the software development process will be described. In this section, we are concerned with what are the most difficult modeling problems that will be encountered, and how the simulation model can be used to support the software development researcher and manager.

Complexity, productivity, and quality are, by far, the most difficult aspects of the software development process for the researcher to understand and for the manager to manage. Complexity and quality are attributes of the software system that is to be developed, while productivity is an attribute of the personnel resources to be applied in the software development. Complexity is important because, along with system size, it gives a measure of the amount of effort required to develop the system. The question is how to quantify system complexity, and then how to represent it so that interdependencies with other system attributes are modeled. Productivity is important because, with a given personnel allocation, we can use productivity to predict project schedules. Lines of code has traditionally been used to express programmer productivity, but the complexity and quality attributes of the code will also affect productivity. Quality is important because the level of quality of an intermediate software product will affect the amount of effort that is required in a subsequent phase of the software development. What is needed are modeling concepts that will allow us to combine the factors of complexity, productivity, and quality in one integrated model.

A simulation model of the software development process may be used for planning and analysis functions, or for management and control functions. Planning and analysis functions include cost and schedule estimation, productivity analysis, and quality analysis. These functions are concerned with how different development and management methodologies will affect cost/schedule, productivity, and quality. Management and control functions include project status estimation and problem identification. These functions are concerned with how to relate project resource consumption to project progress, and how to recognize potential bottlenecks.

The adequacy of the simulation approach for modeling the software development process is summarized in the following table.

Criteria	Evaluation
applicability	excellent
confidence	promising
completeness	excellent
minimality	excellent
independence	excellent

Applicability is excellent because a simulation model can be oriented toward studying any aspect of the software development process. Confidence is promising because if the accuracy of part of the model is not sufficient, then that part of the model can be expanded to a higher level of detail. Completeness is excellent because of the microscopic view that is taken with the simulation approach. Minimality and independence are both excellent because the flexibility inherent in the simulation approach allows system states to be combined or decomposed at the discretion of the modeler.

SECTION 3

DEVELOPMENT OF SIMULATION MODEL

3.1 ORIENTATION OF THE SIMULATION MODEL

Our study of the software development process is motivated by three basic objectives. First, we wish to formulate a model of the software development process to help us to better understand the internal behavior of the process. All of the models that have been developed thus far are macroscopic models which do not provide insight regarding interdependencies between system parameters. We want to develop a microscopic model which will provide the opportunity to explore the internal characteristics of a software development, and the relationships between these characteristics.

Second, we wish to apply simulation techniques to evaluate the validity of our microscopic model with respect to historical software data. Clearly, analytic techniques are not capable of producing the desired microscopic model, so we want to adapt the relevant simulation modeling techniques to represent the software development process on a microscopic level.

Third, we wish to develop a simulation-based methodology for software cost and schedule estimation. Given a validated simulation model of the software development process where the interdependencies between microscopic system characteristics are represented, we want to determine the kinds of system characteristics that should be input to the model. We also want to determine the kinds of behavior statistics of the software development system that should be output to the model user.

3.2 LEVEL OF MODEL DETAIL

It was decided to take a microscopic view of the software development process in order to better understand its internal behavior. We need to select the proper level of model detail. The proper level of detail will depend both on what is needed to satisfy the goals of the simulation study, and on what is practical to implement or collect data for. The objects or entities that will be represented in our model are: people, resources, and products. Let us now examine each of these entities in more detail to determine a suitable level of modeling detail for each entity.

People are the primary entity in our system. There are a number of different types of personnel that are required in a software development. We could discriminate personnel type by job title, or by generic job function. We have chosen the latter approach for simplicity, and because there is no present need for the more detailed distinction. For each generic job function, we can identify a number of <u>activities</u> that are performed by a person having that job function. For example, the activities performed by the system designer are: 1) hardware architecture and configuration; 2) software architecture and decomposition; 3) system performance/reliability analysis; and, 4) system test plan. The level of detail in the specificaiton of activities for each job function remains to be determined. Experimentation with the model will no doubt lead to many refinements in the model. It is anticipated that a multi-level model will result; that is, one with varying levels of model detail for different components of the model.

Resources are the equipment used by software development personnel in order to perform their job function and comprise the second type of entity in our system. Resources may include computers, computer terminals, and keypunch machines. We are primarily interested in resources that affect productivity, or contribute to delays caused by contention. An example of a resource that affects productivity is the computer terminal used in an interactive software development effort. The programmer in an interactive development environment is more productive than one in a batch development environment using keypunch machines. The computer terminal is also an example of a resource that can cause delays due to contention. The level of model detail in our representation of the resources utilized in a software development is thus determined by a simple rule: include only those resources which have a significant impact on project progress.

Software products, both intermediate and final products, are the third type of entity in our system. A software product can be a document, for example, a system requirements specification, or it can be a piece of code, for example, a computer program or subprogram. Software products provide a measure of the progress of a software development in the same way that the utilization of personnel and non-personnel resources is a measure of project progress. The identification of software products enables us to more meaningfully decompose job functions into detailed activities. The level of detail to be used in modeling software products thus depends on the level of detail to which personnel activities are modeled. If activities are modeled in more detail, then a greater number of intermediate software products will be represented in the model.

3.3 MODEL FORMULATION

The software development process has often been described in terms of a sequence of phases: the requirements analysis phase, the preliminary design phase, the detailed design phase, the programming phase, and the integration/test phase. This is a natural way of viewing the software development process, but is of limited use because the definition of these phases is tied to the passing of time, and not to the dynamic behavior of the process itself. This view of the software development process is a milestone-driven view. There are other ways of looking at the software development process that are similar to the phase approach, but not defined in terms of time. These are: the activity progression model, and the product progression model.

In the activity progression model, the software development process is represented by a mix of ongoing development activities. The activities can be measured by the quality, quantity, and type of development activity. An activity is related to the amount of effort required to perform that activity rather than the time to perform it. The types of activity are the activities described in the previous section, the results of decomposing the phases.

The important difference with the activity progression model is that we can have varying mixes of activities ongoing in parallel. For example, during the beginning of the project, there is mostly analysis-oriented activities going on, some design, but no programming. The quantity of development activity is measured by the number of person hours per week that are committed to each activity. The quality of the work that is being performed can be related to the background and experience of the personnel performing the activity. This can be expressed in quantitative terms by assigning quality ratings to various types of background, and to various levels of experience. These concepts are illustrated below.

SYSTEM DEVELOPMENT PROGRESS IS REPRESENTED BY THE MIX OF ONGOING DEVELOPMENT ACTIVITY (WORK BREAKDOWN STRUCTURE MODEL):

TYPE	QUANTITY	QUALITY
REQUIREMENTS ANALYSIS DESIGN CODE AND DEBUG . . .	PERSON HOURS BY TYPE OF PERSON	PERSONNEL EXPERIENCE

Figure 3-1 Activity Progression Model Concepts

The activity progression model is closely related to the work breakdown structure (WBS) concept. In developing a work breakdown structure for a particular software development project, the project is decomposed into a sequence of work packages such that the mix of activities to be performed for each work package is well defined, and such that the quantity of personnel resources required for each work package is easily estimated. The activities in the activity progression model correspond to the work packages in the WBS. There are two possible modes of operation for the activity progression model. For planning purposes, the simulator will pick a Q value for each activity which determines the anticipated experience of the personnel resources to be applied in that activity. The Q value will then be transformed into the effort E required to perform the activity. This transformation will be done by using empirically derived rules. Based on the quantity of personnel resources to be applied in each activity, the simulator will use the derived effort to develop schedule and cost estimates for that activity. For control purposes, the Q value for each activity will be estimated based on the personnel actually being applied. The simulator will use this data to estimate the remaining effort required, from which updated schedule and cost predictions can be made. Figure 3-2 describes the operation of the activity progression model.

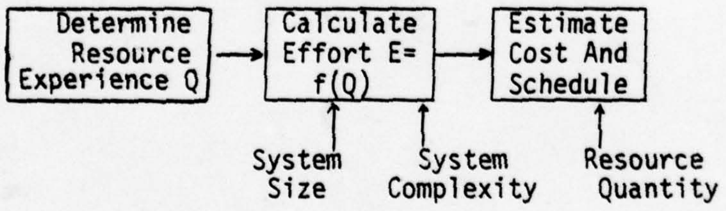


Figure 3-2 Activity Progression Model Processes

In the product progression model, software development progress is represented by the degree of development of system knowledge. In this model, we view the software development process as a process of creative knowledge synthesis. System knowledge starts out in the form of requirements, and evolves through design concepts to program code. The evolution of system knowledge is constrained by technological and economic limitation and, affected by the application of management and development methodologies. We can measure the degree of development of system knowledge by the quality, quantity, and type of each intermediate system product that is produced. The type of intermediate system products will be documents or code. Examples of system products are: system specification, preliminary design document, detailed design document, source code, and test plan. The quantity of a system product can be measured by the number of pages in the document and by the number of lines of source code in the program. The quality of a system product will be expressed by software quality metrics which rate the quality characteristics of the document or code in quantitative terms. These concepts are illustrated in Figure 3-3.

SYSTEM KNOWLEDGE EVOLVES IN THE FORM OF THE PROGRESSION OF PRODUCTS DEVELOPED (PRODUCT PROGRESSION MODEL).

TYPE	QUANTITY	QUALITY
REQUIREMENTS SPEC DESIGN SPEC TEST PLAN SOURCE CODE . . .	LINES OF DOCUMENTATION OR CODE	AS MEASURED BY METRICS

Figure 3-3 Product Progression Model Concepts

The product progression model is closely related to the concept of system knowledge evolution. In modeling the evolution of system knowledge, the only tangible evidence of the changing character of system knowledge is in the intermediate system products. There are two possible modes of operation for the product progression model. For planning purposes, the simulator will pick a Q value for each intermediate system product which determines the anticipated quality of that system product. The Q value will then be transformed into the effort E required to produce the system product. This transformation will be done by using empirically derived rules. Based on the quantity of system product to be produced, the simulator will use the derived effort to develop schedule and cost estimates for each product. For control purposes, the Q value for each system product will be estimated based on the intermediate system products actually produced to that period of time. The simulator will use this data to estimate the remaining effort required, from which updated schedule and cost predictions can be made. Figure 3-4 describes the operation of the product progression model.

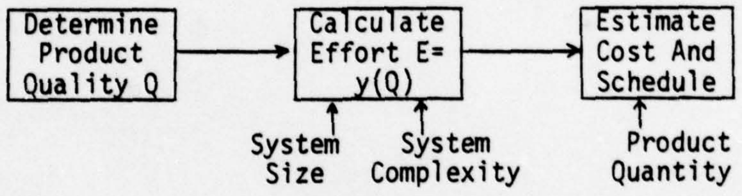


Figure 3-4 Product Progression Model Processes

The activity progression and product progression models are two alternative ways of looking at the software development process. With the activity progression model, we are looking only at activities and quality characteristics of the resources being applied, assuming that products of "standard" quality are produced. With the product progression model, we are looking only at system products and their quality characteristics, assuming that personnel resources of "standard" experience are applied. In general, we must be concerned with both the experience of the resources applied, as well as the quality of the products produced.

Figure 3-5 describes the operation of a combined activity/product progression model:

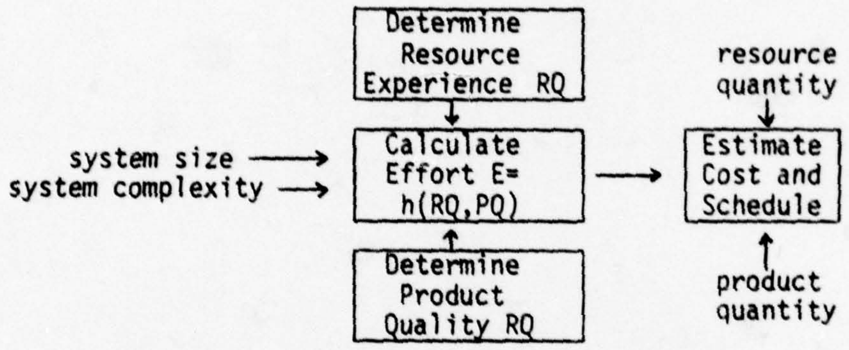

Determine Resource Experience RQ
system size
system complexity
Calculate Effort E= h(RQ,PQ)
Determine Product Quality RQ
resource quantity
Estimate Cost and Schedule
product quantity

Figure 3-5
Combined Activity/Product Progression Model

We have discussed alternative approaches to viewing the software development process -- as a sequence of activities performed, as a sequence of products produced, or both. There are two additional topics that must be addressed as part of our model formulation. One topic is the identification of all the <u>state variables</u> by which the internal and external characteristics of the software development process will be represented. The second topic is the identification of all the <u>analytic and algorithmic rules</u> by which the dynamic behavior of the software development process will be represented. The analytic rules typically represent the time dependent behavior of the state variables, or the interdependencies between state variables. The algorithmic rules typically represent the conditions for when to apply an analytic or another algorithmic rule, or when to change the nature of an analytic or algorithmic rule. Algorithmic rules also specify the time dependent grouping of analytic or other algorithmic rules.

The state variables of the model of the software development process can be classified into three groups -- first, those variables describing the software system to be developed; second, those variables describing the personnel and computer resources to be applied during the project; and, third, those variables which describe the progress of the software development. State variables describing the software system include system size and system complexity. State variables describing the project resources include number and experience level of each type of personnel resource, and number of each computer resource. State variables describing project progress include active/inactive flags for all activities and products, and cumulative effort expended on each activity and product. Further analysis remains to be performed to develop a more complete set of system state variables.

The rules that represent the dynamic behavior of the software development process include analytic as well as algorithmic rules. Analytic rules can be used to express the effort required to perform a certain activity or to produce a certain software product as a function of type of activity or product, and quantity and quality of activity or product. Analytic rules can also be used to express time and dollar costs for a given activity or product as a function of the effort required. Algorithmic rules are normally used to represent the interdependence between activities or between products. For example, when the cumulative effort being applied on a particular activity or product reaches a threshold, then that activity or product is completed, and subsequent effort applies toward the completion of the next activity or product. In our preliminary analysis we have attempted only to establish the kinds of analytic and algorithmic rules to be included in a model of the software development process. A more detailed and complete set of rules remains to be established, and empirical studies need to be performed to calibrate and validate the rules that are developed.

3.4 MODEL INPUTS

We distinguish between two categories of simulation model inputs -- the model parameter and the model calibration values. The model parameter values are the values of the system state variables which describe the system characteristics, the resource characteristics, and the project characteristics. The model calibration values represent those factors which affect the behavior of the model which are not system state variables. Each software development environment will have its own set of model calibration values. Each software development environment can be applied to developing different software systems with different personnel resources. The factors which affect the model calibration values are the management methodology, which includes how software development teams are structured, and the development methodology, which includes what software development tools are available for use by project personnel. Some of the anticipated inputs are shown in Table 3-1.

MODEL PARAMETERS

- SIZE
  - by subsystem
  - lines of source code
  - type of language
  - percent new code
- COMPLEXITY
- TYPE OF SYSTEM
- CONSTRAINTS
  - processing time
  - storage size

CALIBRATION PARAMETERS

- DEVELOPMENT MANAGEMENT PLAN
  - methodology
  - organization
  - staff mix
  - milestones
  - products/quality goals
  - development tools
  - perceived risk areas
- ENVIRONMENTAL FACTORS

Table 3-1
Anticipated Model Inputs

3.5 MODEL OUTPUTS

Several modes of operation are anticipated for a simulator. The usual mode would be to provide the simulator the required schedule and the input data identified in Table 3-1. The output would be a resource expenditure profile and the cost. Another mode of operation would be to input a resource profile and have the simulator provide schedule and cost information. A third mode of operation is to provide cost data, and allow the simulator to produce a resource profile and schedule. The schedule and cost information could be generated for the entire software development project, or can be generated by activity or by product. The information that is generated can be static information, that is, summary statistics of the values of selected system state variables, or histograms of these state variables. On the other hand, the information that is generated can be dynamic, that is, a trace of the values of system state variables at each value change, or a plot of these state variables against time as the independent variable. The execution of the simulation model represents the behavior of the software development process: we can make one execution and collect statistics which indicate the model's static and dynamic characteristics. Alternatively, we can make several executions, and compare the behavior of each model to arrive at an optimal schedule or resource allocation plan.

In addition to this standard output, output that supports risk and sensitivity analyses and projections of anticipated maintenance levels will be available. These outputs are shown in Table 3-2.

SUMMARY	DETAILED
• COST	• ACTIVITY PERFORMANCE
• SCHEDULE	• MILESTONE PERFORMANCE
• RESOURCE EXPENDITURE PROFILE	• RESOURCE UTILITZATION
• RISK	• RESOURCE QUEUE STATISTICS
• SENSITIVITIES	• DYNAMIC SIMULATION TRACE
• PROJECT MAINTENANCE	

Table 3-2 Anticipated Model Outputs

SECTION 4

DECOMPOSITION OF THE SOFTWARE DEVELOPMENT PROCESS

4.1 INITIAL CONCEPTS

The challenges of managing a software development are immense because it is a multi-element process which is highly coupled and highly complex, and there are wide variations in controllable and uncontrollable variables between projects. In the past, the technique used by managers has been to decompose the software development process into "independent" subprocesses and manage those separately. This technique, generally following the Wolverton description [WOLV 72], does not usually reflect a very accurate model of the way software is currently being developed or is not in enough detail to analyze the causes of poor performance [TURN 76]. The "waterfall" diagram, (Figure 4-1), in fact, originally was utilized to project the idea that each phase should be completed and validated before beginning the next phase [CARR 75]. There has been recognition in recent years that interaction occurs, and that a continuous configuration management effort is required to keep the products of each phase up to date and consistent. Thus, the traditional widely used "model" of the software development process is outmoded, no longer representing a true picture of how software is developed.

Another reason for a more detailed model of the software development process is to assist in estimating the effort and cost of a project. The following quotes provide the opinions of several researchers in this field:

> Estimates will improve only when the estimators achieve greater insight and understanding of the system development process, and the functions which make up the process, of the interdependencies of these functions and of the factors which influence the resource requirements of the functions. [GEHR 76]

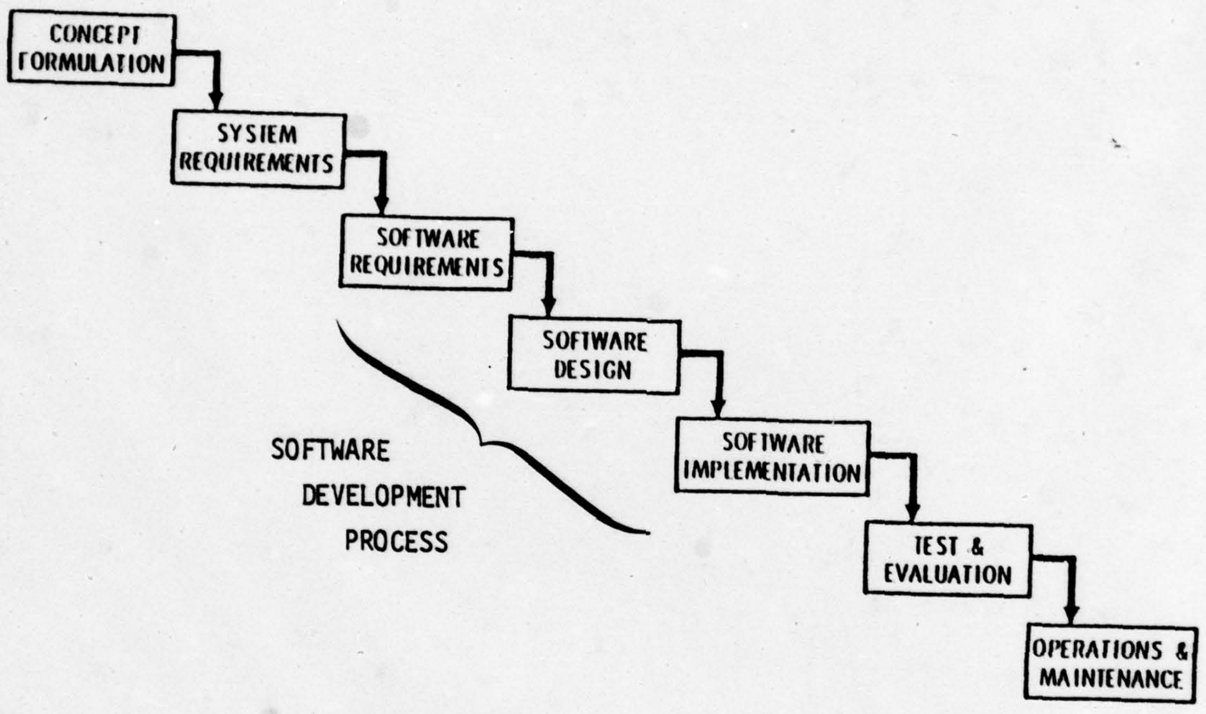


Figure 4-1 Traditional Software Development Process Model

Many of the problems of resource estimating are symptoms of an underlying ignorance of the process of program system development for which the estimates are being made. The serious students of estimating must first be willing to probe deeply into the fascinating and complex software development process; to uncover the phases and functions of the process; to highlight the subtle interrelationships of the program system being developed and the project organization doing the developing.

Those who persevere, however, will recognize that examining the influencing variables and their causal relationships is precisely what is required if estimates are ever to be improved. Only then can we do meaningful quantitative research and scientific analysis of resource requirements. We are never likely to eliminate unpredictable variability, but we should be able to go a long way toward improving predictability far above today's primitive state-of-the-art. [PIET 7C]

At present, most managers focus on only one of the process inputs -- labor -- and one of the outputs -- lines of code. [KOLA 76]

This detail will not only assist in estimating the task, but also in assessing risk, allocating resources, and planning strategies; i.e., developing a detailed development plan. It will provide better visibility of the assumptions and dependencies upon which the success or failure of the development effort depends.

The advantages or benefits that could be derived from a more realistic model of the software development process are:

- A model which describes the process in more breadth and detail to provide a better understanding of the interactions and relationships within the process
- A perspective to evaluate and analyze the symptomatic data which are gathered during a development
- A framework in which the utility of new tools and methodologies can be evaluated.
- A planning aid to assist in establishing resource mix require-

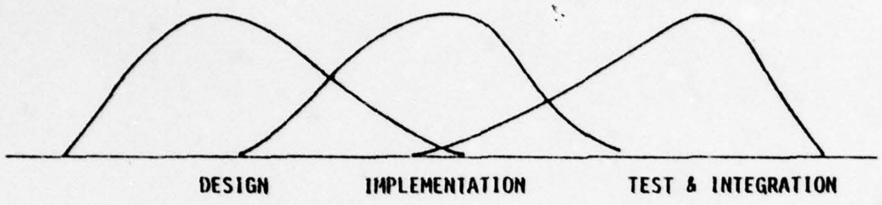
ments and schedules

- A project management aid to enhance progress determination and problem identification

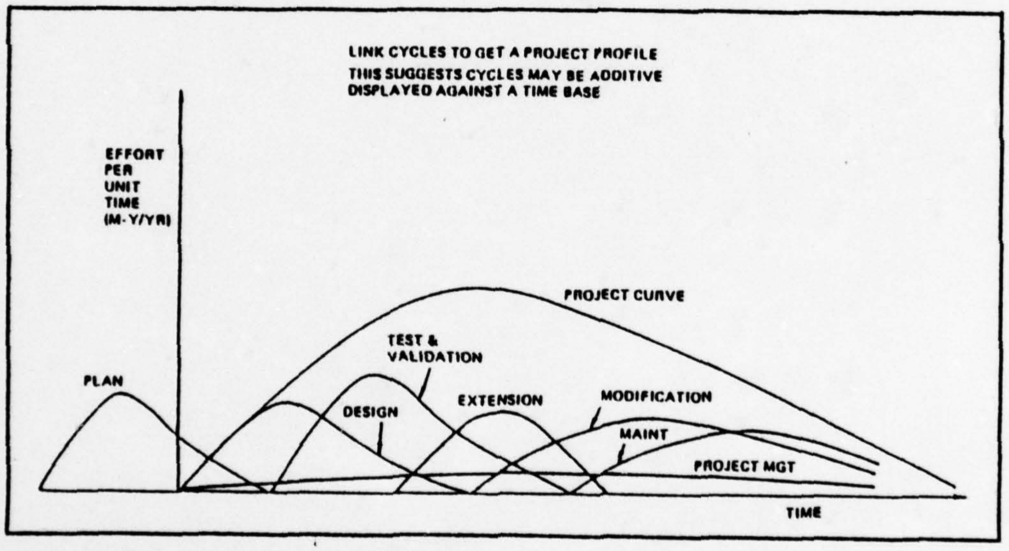
Current cost estimation techniques ([PUTN 77b]), [REFE 77]) do not provide any greater detail in their model of the software development process, as shown in Figure 4-2.

The approach taken in this study was to attempt to model how software is actually developed. As a starting point, an extension to Wolverton's model that has been recognized in recent years is the addition of feedback paths as shown in Figure 4-3. This model acknowledges that redesign, revisions to the requirements, and changes to the source code take place constantly in the development of a software system. This iteration not only relates to the correction of problems found in the later stages of the development, but also represents an increase in the amount of knowledge about the system -- its functions and uses -- that takes place over the time span of a development.

These corrections and revisions take place as a function of the activities the development personnel perform, not as a complete recycle of a phase, as depicted in Figure 4-3. Therefore, Figure 4-4 is a representation of this growth of knowledge about a system on a timeline. A part of each subsequent phase to requirements analysis is correcting, modifying, or adding to the requirements stated at the end of the requirements analysis phase. This increase or expansion of detail, knowledge, and documentation about the system that occurs during succeeding phases, then, includes the updates and revisions to the preceeding concepts. These concepts were previously discussed in Section 3.



PRICE-S
EXPENDITURE
PROFILE BY PHASE



PUTNAM LIFE
CYCLE MODEL
EXPENDITURE
PROFILE BY
PHASE

Figure 4-2 Current Cost Estimation Technique's Depiction Software Development Process

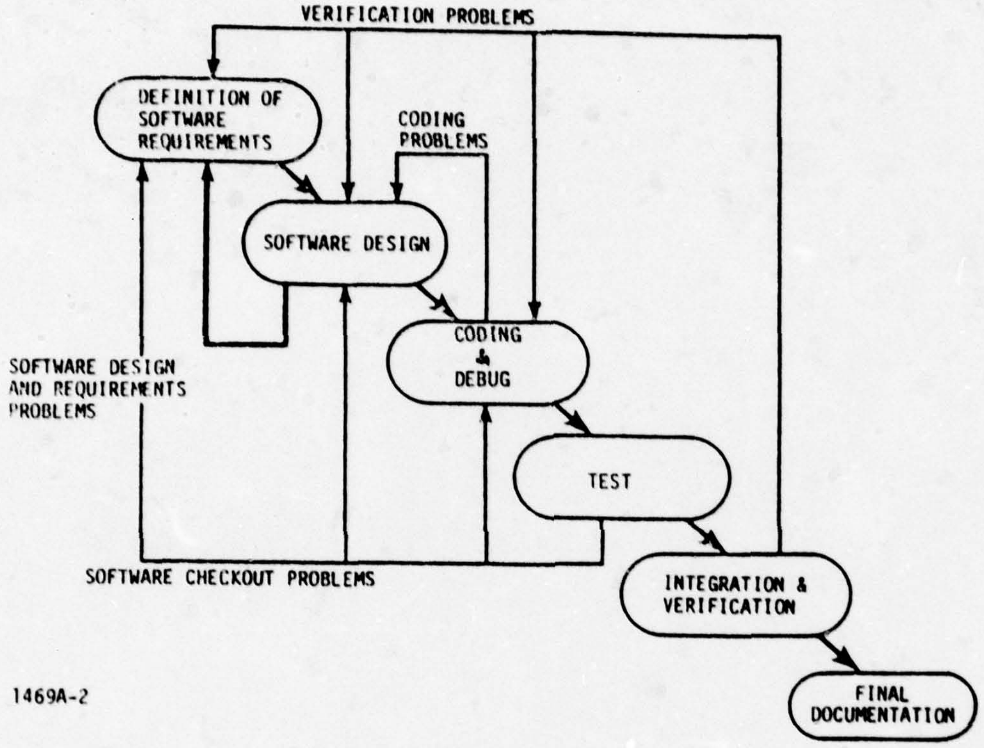
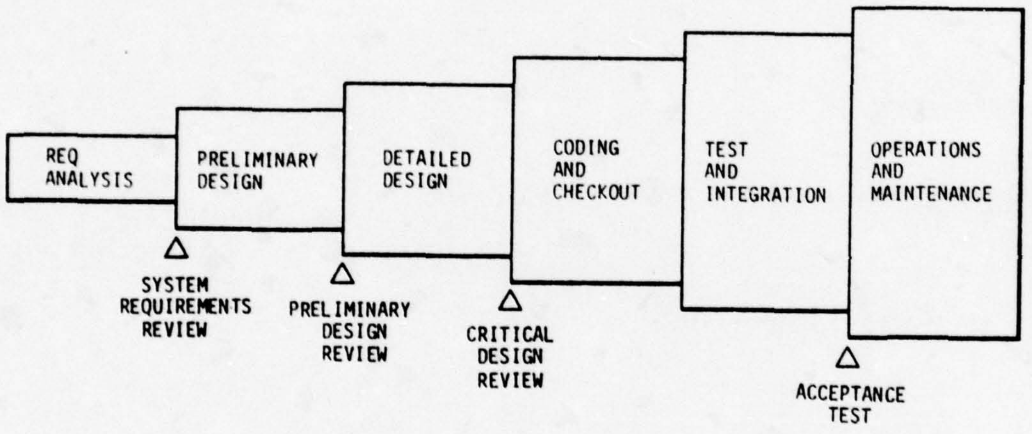

VERIFICATION PROBLEMS
DEFINITION OF SOFTWARE REQUIREMENTS
CODING PROBLEMS
SOFTWARE DESIGN
CODING & DEBUG
SOFTWARE DESIGN AND REQUIREMENTS PROBLEMS
TEST
SOFTWARE CHECKOUT PROBLEMS
INTEGRATION & VERIFICATION
1469A-2
FINAL DOCUMENTATION

Figure 4-3 Software Development Process with Feedback Paths



1775A-1

Figure 4-4 Timeline Representation of Software Development Process

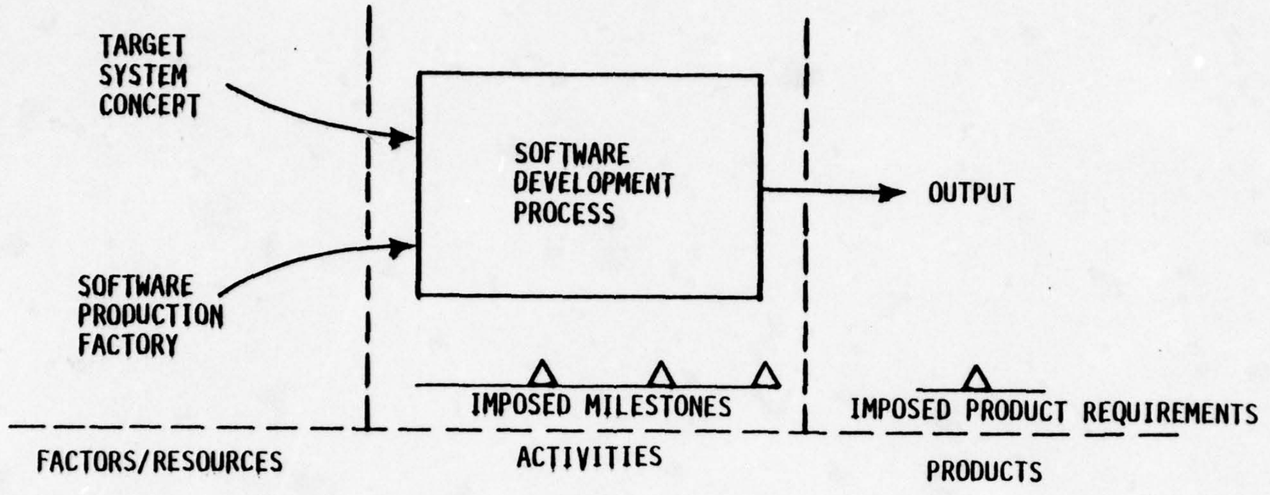
4.2 APPROACH TO DECOMPOSING THE SOFTWARE DEVELOPMENT PROCESS

Conceptually, the software development process is a process which is driven by a concept of, or requirement for, a target system and utilizes the resources of a software production factory to produce an operational system. The target system is a software system which has certain desired characteristics. These characteristics have an impact on the amount of resources which are consumed or utilized in the process of producing the operational system. The software production factory is the organization, staffing, and development strategies superimposed on the resources of a project group (consisting of personnel and development tools), which provide the production capability and environment for accomplishing the system development. The operational system, the output of the process, is represented by the documents, data, and code produced as a result of the software development.

Imposed on this development process are a series of milestones which represent intermediate formal reviews of the progress towards the operational system. Further, there are documentation requirements which define what products are to be delivered. Almost all software developments have these milestone and documentation requirements. Perhaps the most rigorous set of requirements are those imposed by military standards. These concepts are shown in Figure 4-5.

The currently utilized models of the software development process, discussed in Section 4.1, represent a high level decomposition of the process, oriented toward the milestones imposed. Current cost estimation techniques basically attempt to replace the process with a relationship that represents the tranformation from the inputs, represented by the target system characteristics and the resources of the software production factory, to the outputs, represented by the documentation and code.

Our approach to modelling the software development process was to


TARGET SYSTEM CONCEPT
SOFTWARE PRODUCTION FACTORY
SOFTWARE DEVELOPMENT PROCESS
OUTPUT
IMPOSED MILESTONES
IMPOSED PRODUCT REQUIREMENTS
FACTORS/RESOURCES
ACTIVITIES
PRODUCTS

1781A

Figure 4-5 Software Development Process Concept

decompose the process in more detail. The methodology used to accomplish this decomposition involved a three-dimensional view (as shown in Table 4-1): identification of the products of the software development process; identification of the activities that comprise the process; and, identification of the factors and resources that represent the target system and the software production factory.

The products of this process that have been identified are those required by militray standards. Intermediate products, as well as final products, were considered. In addition to the number, type, and size of the products and the resources required to produce them, the quality of the product was considered. The effort required to update and modify the documents during subsequent activities is affected by the initial quality. Taking this phenomena into account, we hope to be able to model the interactions between phases that current cost estimation techniques are unable to model [GRAV 76].

Identification of the various activities in the process was accomplished by decomposing each phase identified in the high level models into the typical activities performed during that phase. The activities were identified at a level of detail at which we feel valid simulation models can be developed, and at which valid empirical data can be collected. Further refinement could, and may, be done if experimentation indicates that it is necessary. However, it is felt that refinement to a level where individual decisions are modelled and individual personnel within the development process are modelled introduces far too much variation for simulation techniques to be truly effective. The level to which the activities identified are modelled is the team level; i.e., a group of people working on a particular aspect of the software system at a particular time in the development. The interdependencies between the activities and the types of resources typically utilized were also identified.

TABLE 4-1

DECOMPOSITION METHODOLOGY

(1) Identify Products (intermediate and final)
- type, size
- resource allocation
- quality

(2) Identify Activities
- decompose phases
- resource utilization
- relationship to products
- interdependencies of activities

(3) Identify Factors and Resources
- impact on activities

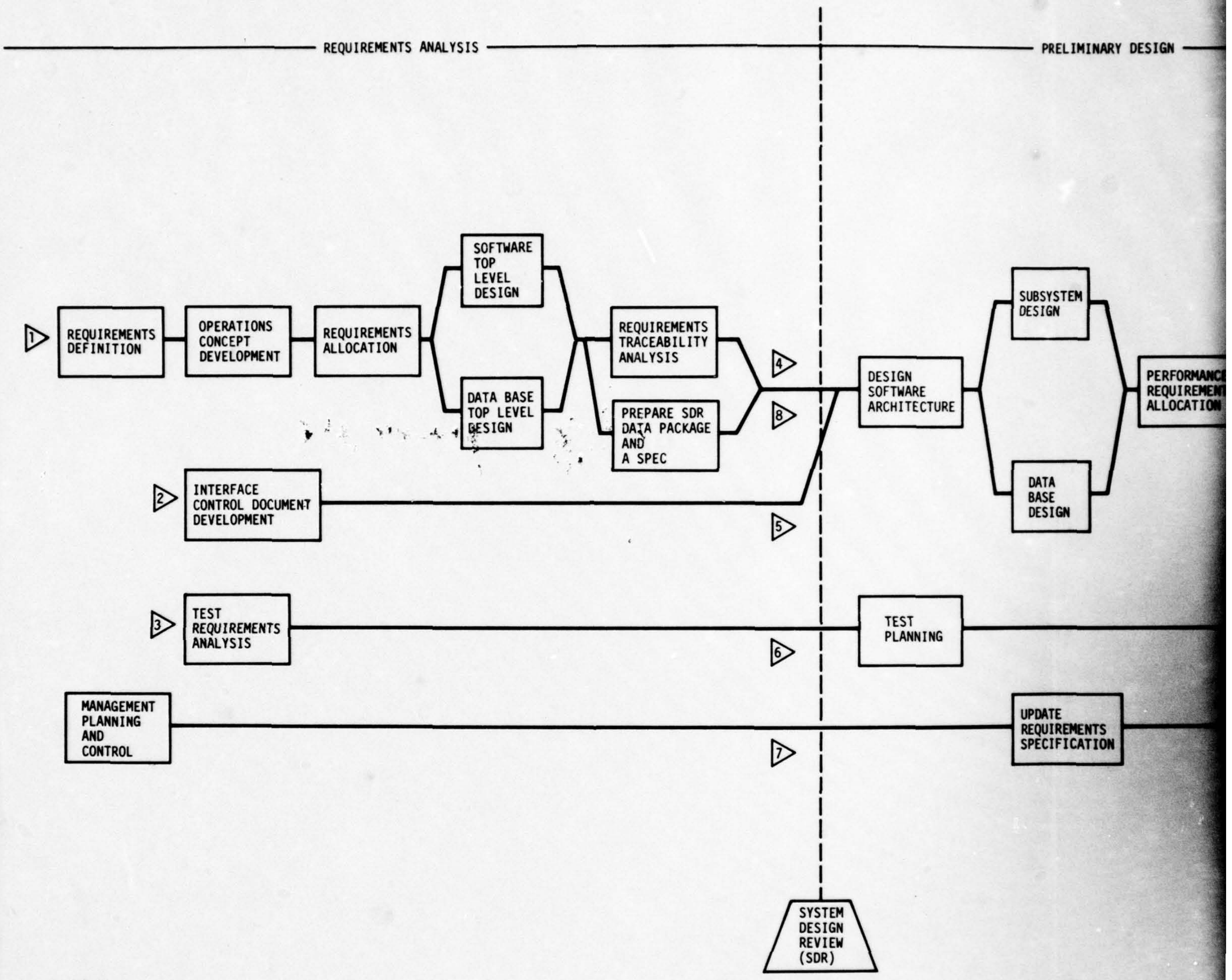
The last aspect of the decomposition involved identifying how the resources, target system, and factors affecting the amount of resources required would be described. These descriptive factors are the core of the models and the input variables to the simulator. The results of applying this methodology are contained in the next paragraph.

4.3 DESCRIPTION OF MODEL

The activities, their interrelationships, and the products of the Software Development Process are shown in Figure 4-6. This network of activities represents a typical large-scale software development. While our intent was to identify generic activities, the ones identified are biased by software developments performed in accordance with military standards. Our design approach, which is illustrated in the next section, where a prototype simulator is described, involves a library of these activities from which a subset could be chosen and interconnected to represent a particular development. Other activities could be identified. Table 4-2 provides an explanation of each of the activities identified in the figure.

Note that in Figure 4-6 several activities in the detailed design phase and implementation phase are duplicated. These activities are performed on a subsystem, or CPCI level. In the figure, a two-subsystem development is represented. The sequence of activities, or process thread, which is duplicated, would be repeated for as many subsystems as identified in the target system. Each of these process threads are performed normally by a specific group, in this case a programming team, in the development organization. Other process threads can be recognized. in the model description. For example, the process thread which begins with Test Requirements Analysis in the Requirements Analysis phase and proceeds through Test Planning, Test Planning and Preparation, System Test Cast Generation, System Test Data Generation, and ends with System Tests and Acceptance Tests would normally be performed by an independent quality assurance or test group. Figure 4-6, then, provides a phase/ activity decomposition of the software development process illustrating process flow and data or product flow.

The resources that are typically utilized in a software development are identified in Table 4-3. The phase in which they are used is also shown. Our model to date, represented by the prototype, only considers a subset


REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
1
REQUIREMENTS DEFINITION
OPERATIONS CONCEPT DEVELOPMENT
REQUIREMENTS ALLOCATION
SOFTWARE TOP LEVEL DESIGN
DATA BASE TOP LEVEL DESIGN
REQUIREMENTS TRACEABILITY ANALYSIS
PREPARE SDR DATA PACKAGE AND A SPEC
4
8
DESIGN SOFTWARE ARCHITECTURE
SUBSYSTEM DESIGN
DATA BASE DESIGN
PERFORMANC
REQUIREMENT
ALLOCATION
2
INTERFACE CONTROL DOCUMENT DEVELOPMENT
5
3
TEST REQUIREMENTS ANALYSIS
6
TEST PLANNING
MANAGEMENT PLANNING AND CONTROL
7
UPDATE REQUIREMENTS SPECIFICATION
SYSTEM DESIGN REVIEW (SDR)

INPUTS

▷1 SYSTEM SPECIFICATION (A SPEC)

▷1 SYSTEM REQUIREMENTS REVIEW DATA PACKAGE

▷1 SEGMENT SPECIFICATION

▷2 SYSTEM INTERFACE CONTROL DOCUMENT

▷3 SYSTEM INTEGRATED TEST PLAN

PRODUCTS

▷4 SOFTWARE SYSTEM REQUIREMENTS SPECIFICATION (A SPEC)

▷5 INTERFACE CONTROL DOCUMENT

▷6 TEST REQUIREMENTS SPECIFICATION

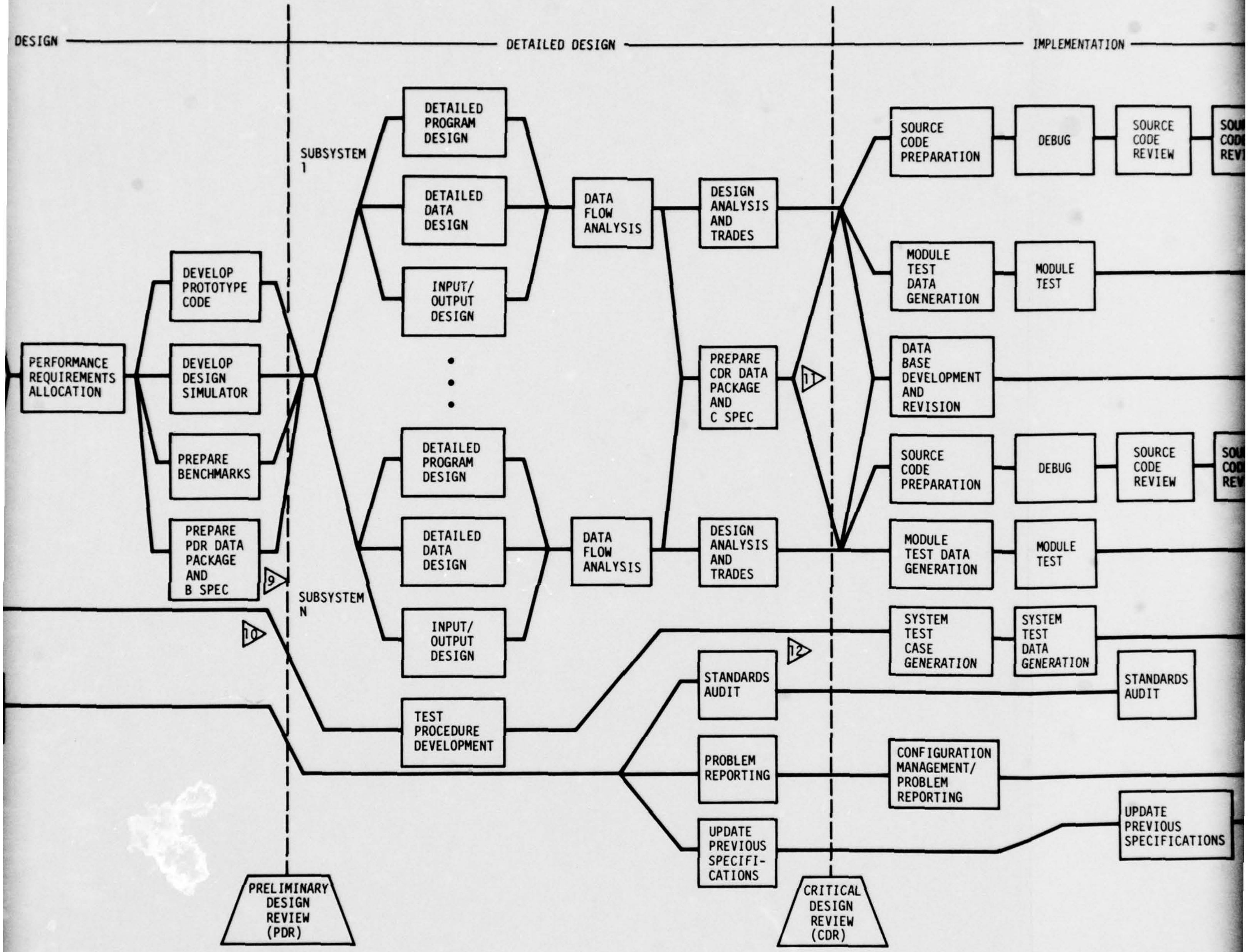
▷7 QUALITY ASSURANCE PLAN

▷7 MANAGEMENT PLAN

▷7 CONFIGURATION MANAGEMENT PLAN

▷7 DEVELOPMENT STANDARDS AND CONVENTIONS

▷8 SYSTEM DESIGN REVIEW DATA PACKAGE


DESIGN
DETAILED DESIGN
IMPLEMENTATION
PERFORMANCE REQUIREMENTS ALLOCATION
DEVELOP PROTOTYPE CODE
DEVELOP DESIGN SIMULATOR
PREPARE BENCHMARKS
PREPARE PDR DATA PACKAGE AND B SPEC
9
10
SUBSYSTEM 1
SUBSYSTEM N
DETAILED PROGRAM DESIGN
DETAILED DATA DESIGN
INPUT/ OUTPUT DESIGN
DETAILED PROGRAM DESIGN
DETAILED DATA DESIGN
INPUT/ OUTPUT DESIGN
TEST PROCEDURE DEVELOPMENT
DATA FLOW ANALYSIS
DATA FLOW ANALYSIS
DESIGN ANALYSIS AND TRADES
PREPARE CDR DATA PACKAGE AND C SPEC
DESIGN ANALYSIS AND TRADES
STANDARDS AUDIT
PROBLEM REPORTING
UPDATE PREVIOUS SPECIFI- CATIONS
11
12
SOURCE CODE PREPARATION
DEBUG
SOURCE CODE REVIEW
SOU CODE REVI
MODULE TEST DATA GENERATION
MODULE TEST
DATA BASE DEVELOPMENT AND REVISION
SOURCE CODE PREPARATION
DEBUG
SOURCE CODE REVIEW
SOU COD REV
MODULE TEST DATA GENERATION
MODULE TEST
SYSTEM TEST CASE GENERATION
SYSTEM TEST DATA GENERATION
STANDARDS AUDIT
CONFIGURATION MANAGEMENT/ PROBLEM REPORTING
UPDATE PREVIOUS SPECIFICATIONS
PRELIMINARY DESIGN REVIEW (PDR)
CRITICAL DESIGN REVIEW (CDR)

PRODUCTS

9 COMPUTER PROGRAM REQUIREMENTS SPECIFICATION (B SPEC)

9 PRELIMINARY USERS MANUAL

9 FINAL INTERFACE CONTROL DOCUMENT

9 DATA BASE SPECIFICATION

9 PRELIMINARY DESIGN REVIEW DATA PACKAGE

10 TEST PLAN

PRODUCTS

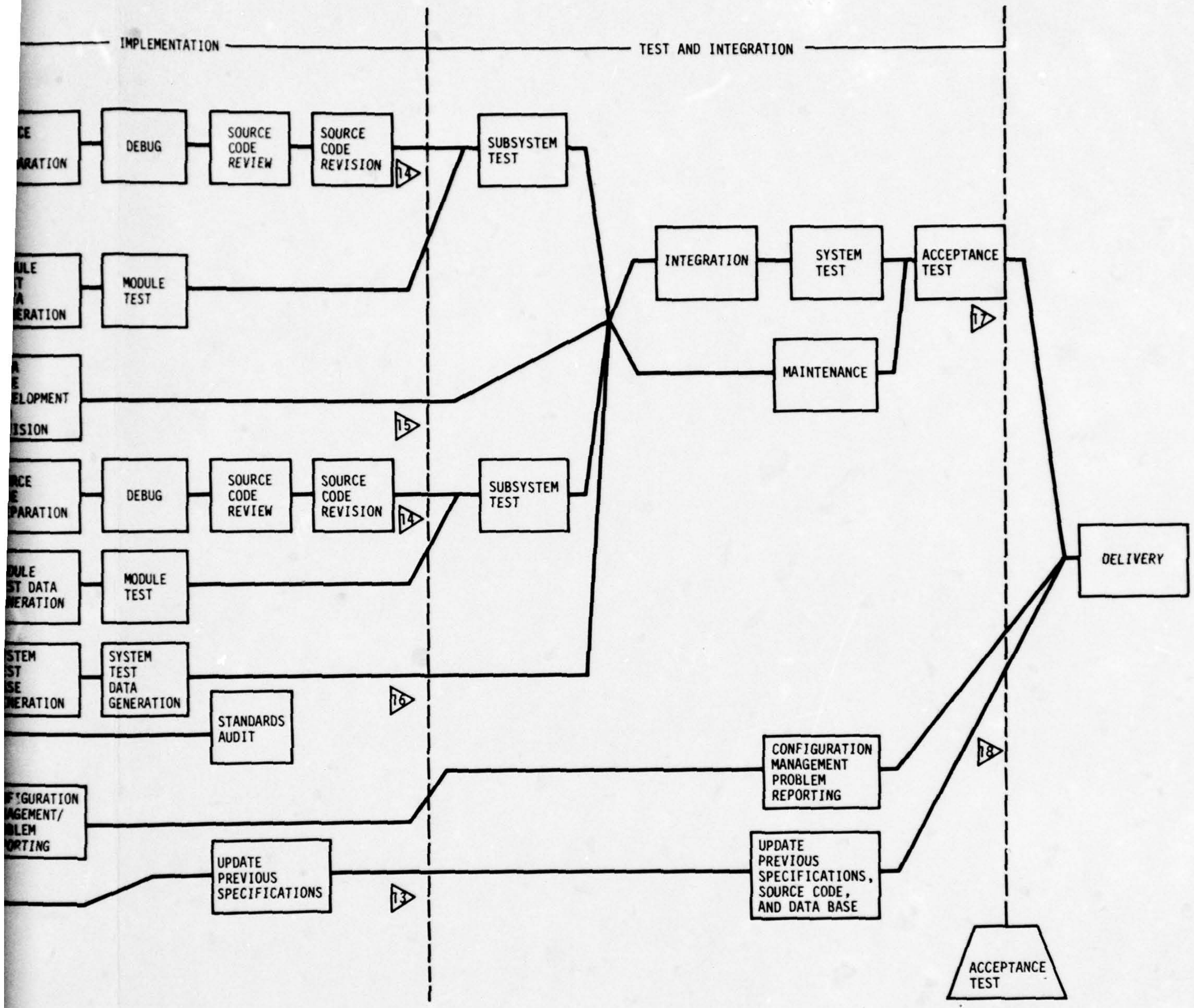
11 COMPUTER PROGRAM DESIGN SPECIFICATION (C SPEC, BUILD TO)

11 DATA BASE SPECIFICATION

11 CRITICAL DESIGN REVIEW DATA PACKAGE

12 TEST PROCEDURES

2



PRODUCTS

13	COMPUTER PROGRAM DESIGN SPECIFICATION (C SPEC, BUILT TO)
13	USERS MANUAL
14	CODE, LISTINGS
15	DATA STRUCTURES AND VALUES
16	TEST CASES AND TEST DATA

PRODUCTS

17	TEST RESULT REPORTS
17	MAINTENANCE MANUAL
18	DELIVERED SYSTEM (INCLUDING ALL UPDATED PRODUCTS)

3

FIGURE 4-6 SOFTWARE DEVELOPMENT PROCESS MODEL

Table 4-2 Activity Explanations

<u>REQUIREMENTS ANALYSIS ACTIVITIES</u>

Requirements Definition

- interpret and itemize requirements provided in the system specification

Operations Concept Development

- define the system concept identifying the functions or processes to be performed and their interactions
- define user interface at high level

Requirements Allocation

- allocate the requirements to the functions or processes to be performed
- identify high level performance requirements

Software Top Level Design

- identify which functions are candidates for implementation in software
- develop a hierarchy of those software functions
- identify at a high level the data flow between those functions

Data Base Top Level Design

- identify requirements for major files or data bases

Requirements Traceability Analysis

- assess the requirements allocation and top level design for coverage of itemized requirements

Interface Control Document Development

- define interfaces between major system components or segments

Test Requirements Analysis

- develop preliminary test and integration requirements

Management Planning and Control

- document management, quality assurance, configuration management plans
- establish development standards and conventions

Table 4-2 Activity Explanations (Continued)

Prepare SDR Data Package and A Spec

- document results of the technical requirements analysis activities in form of software system requirements specification (A Spec)
- prepare data package for system design review

System Design Review

- customer review of top level design

Table 4-2 Activity Explanations (Continued)

PRELIMINARY DESIGN ACTIVITIES

Design Software Architecture
- software hierarchy to CI level
- CI interfaces established

Subsystem Design
- process design
- algorithm development
- define system inputs and outputs

Data Base Design
- data base structure established
- data definition

Performance Requirements Allocation
- storage and timing allocations

Develop Design Simulator
- prepare simulation for design analysis

Develop Prototype
- prototype coding for design/concept analysis

Prepare Benchmarks
- prepare benchmarks for performance analysis

Update Requirements Specification
- make appropriate modification to requirements based on SDR and preliminary design activities

Test Planning
- develop plans for assessing the software systems compliance with requirements

Prepare PDR Data Package and B Spec
- document results of technical preliminary design activities in form of preliminary design specification (B Spec)
- prepare data package for preliminary design review (PDR)

Table 4-2 Activity Explanations (Continued)

<u>DETAILED DESIGN ACTIVITIES</u>

Detailed Program Design

- CI and module design
- detailed interface definition
- algorithm detailed design
- representation of design by design charts
- local variable identification

Detailed Data Design

- complete data element definition

Input/Output Design

- detailed design of input/output variables, format, media to be used

Data Flow Analysis

- Trace data flow between modules throughout system

Design Analysis and Trades

- assess design using simulator, prototype code, or benchmarks
- evaluate alternative designs

Standards Audit

- insure design representation complies with standards and conventions

Problem Reporting

- maintain design problem reports
- attend configuration control meetings
- track problem resolution

Update Requirements and Preliminary Design Specifications

- modify specifications based on PDR critique or detailed design activities

Test Planning and Preparation

- develop test procedures for CI and system tests

Table 4-2 Activity Explanations (Continued)

Prepare CDR Data Package and C Spec

- document results of technical detailed design activities in form of computer program design specifications (C Spec)
- prepare critical design review data package

Critical Design Review

- customer review of detailed design

Table 4-2 Activity Explanations (Continued)

IMPLEMENTATION ACTIVITIES

Source Code Preparation

- code design
- code entry (keypunch, interactive)
- compilation/assembly
- prepare users manual

Debug

- location and correction of compilation/assembly errors

Source Code Review

- development team review via code inspection, walkthrough, etc.

Source Code Revision

- revisions to code
- entry of revisions
- recompilation/assembly

Module Test Data Generation

- develop test data for module

Module Tests

- conduct tests on individual modules

Data Base Development and Revision

- develop data structures
- establish data values
- modify as a result of tests

Configuration Management/Problem Reporting

- maintain and control changes to source code
- maintain design and code problem reports
- attend configuration control meetings
- track problem resolution

Standard Audit

- insure source code is prepared according to standards and conventions

Table 4-2 Activity Explanations (Continued)

Update requirements, preliminary design, and detail design specifications

- modify specifications based on activities performed during implementation, problem report resolution, CDR critique

System Test Case Generation

- develop test cases to be used

System Test Data Generation

- develop test data for system testing

Table 4-2 Activity Explanations (Continued)

TEST AND INTEGRATION ACTIVITIES

Subsystem Tests

- perform CPCI tests
- prepare test results

Integration

- perform required builds, compilations, etc. to link code as system

System Tests

- subject software to established system tests
- prepare test results

System Maintenance

- identify and correct problem identified during tests
- reenter code
- recompile
- prepare maintenance manual

Update Requirements, Preliminary Design, Detail Design Specifications, Source Code, and Data Base

- modify specifications, source code, and data base based on results of integration and test activities
- prepare for delivery of system

Configuration Management/Problem Reporting

- maintain and control changes to source code
- attend configuration control meetings
- track problem resolutions

Acceptance Test

- perform acceptance tests

TABLE 4-3

PERSONNEL RESOURCES

RESOURCES	PHASES: REQUIREMENTS ANALYSIS	PRELIMINARY DESIGN	DETAILED DESIGN	IMPLEMENTATION	TEST AND INTERGRATION
SYSTEM ENGINEERS	X	X	X	X	X
SYSTEM ANALYSTS	X	X	X	X	X
QUALITY ASSURANCE ANALYSTS	X	X	X	X	X
TESTERS	X	X	X	X	X
PROGRAMMERS		X	X	X	X
PROGRAM LIBRARIANS			X	X	X
DATA ENTRY/KEYPUNCHERS			X	X	X
OPERATORS			X	X	X
CONFIGURATION MANAGEMENT		X	X	X	X
MANAGEMENT	X	X	X	X	X
ADMINISTRATIVE SUPPORT	X	X	X	X	X
TECHNICAL PUBLICATIONS	X	X	X	X	X
SPECIALISTS such as					
OPERATIONS RESEARCH ANALYSTS	X	X			
HUMAN FACTORS ENGINEERS	X	X			
SOFTWARE SCIENCE SPECIALIST	X	X			
STATISTICANS	X	X			
BEHAVIORAL SCIENTISTS	X	X			
AUDITORS			X	X	X
FACILITES ENGINEERS	X	X			
TRAINING ANALYSTS AND INSTRUCTORS					X
METHODS AND PROCEDURES SPECIALISTS	X	X			
APPLICATIONS SPECIALISTS	X	X			
RELIABILITY ANALYST	X	X			

of these resources. The resources also include equipment such as computers, terminals (TTY, CRT), keypunches, and RJE terminals. The equipment resources are only modelled if contention for them introduces a possible delay in the corresponding activity.

The resources are consumed or utilized at the activity level. This allocation of resources to activities is not shown in Table 4-3, but is done within the simulation model of each activity. The allocation of personnel resources is generally done at a team level. Thus, one input to the model will be the expected composition of the various teams or groups comprising the development organization. A chief programmer team-like organization was modelled in the prototype.

It is anticipated that the resources will be modelled, not only by type, but by productivity levels within the type; i.e., senior and junior level programmers will have different productivity rates. Only one level was modelled in the prototype.

The last aspect of our model of the software development process is the identification of the factors that influence sizing decisions within the activity models. Considerable work has been done in cost estimation, software engineering, and software psychology research oriented toward identifying the factors which influence the effort required to develop software. References are identified in Section 7.

In the model, an activity is a sequence of tasks within a software development that consumes or utilizes resources. The amount of resources, or the effort, required to perform those tasks is a function of the <u>goals</u> of the activity, the <u>size/difficulty</u> of the systems or subsystem being developed, the <u>resources</u> that are available, and the <u>productivity</u> at which those resources perform (personnel) or can be utilized (development tools).

Some of the factors that comprise the goals, size/difficulty, resources, and productivity were identified qualitatively as inputs to the simulator in Table 3-1. It is the intent of this section to describe our concept for modelling their impact quantitatively.

The goals of an activity are related to the products that activity produces or to which the activity contributes, the required quality of those products, and the schedule imposed on the activity. The size/difficulty of the system is represented by the number of subsystems to be developed, their estimated size in lines of source code, the size of the data base to be developed, and the complexity of the application. The complexity is a calculated rating based on the application type, the processing time requirements (real time, on-line, time-constrained, or non-time constrained), the anticipated memory utilization, and the dependency on hardware development. The resources available are the personnel. Their productivity is impacted by their organization, experience, the development tools to be used, and turnaround time on the development computer. These factors are identified in Table 4-4.

The concept for how the factors will be used to model an activity is shown in Figure 4-7. The factors related to the size/difficulty of the task are the input model parameters identified in Table 3-1. They will be combined with the goals of the development (products, quality, and schedule), and utilize product/effort transformation relationships to derive the effort required to accomplish the activity. Schedule impact on the effort required to perform an activity is modelled in these relationships.

The resources available and their organization will be combined and utilize resource productivity figures to derive the expected productive effort achievable. The affect of team structure, basically in the form of communication overhead which negatively impacts productivity, is

Table 4-4

Factors Which Affect Activities

<u>DEVELOPMENT GOALS</u>

Products
- number
- type

Quality
- error rates

Schedule
- milestones

<u>SIZE/DIFFICULTY ESTIMATE</u>

System Size
- number of subsystems (CPCI's)
- number of lines of source code (language type, percent new code)
- data base size (number of data sets, number of preset values)

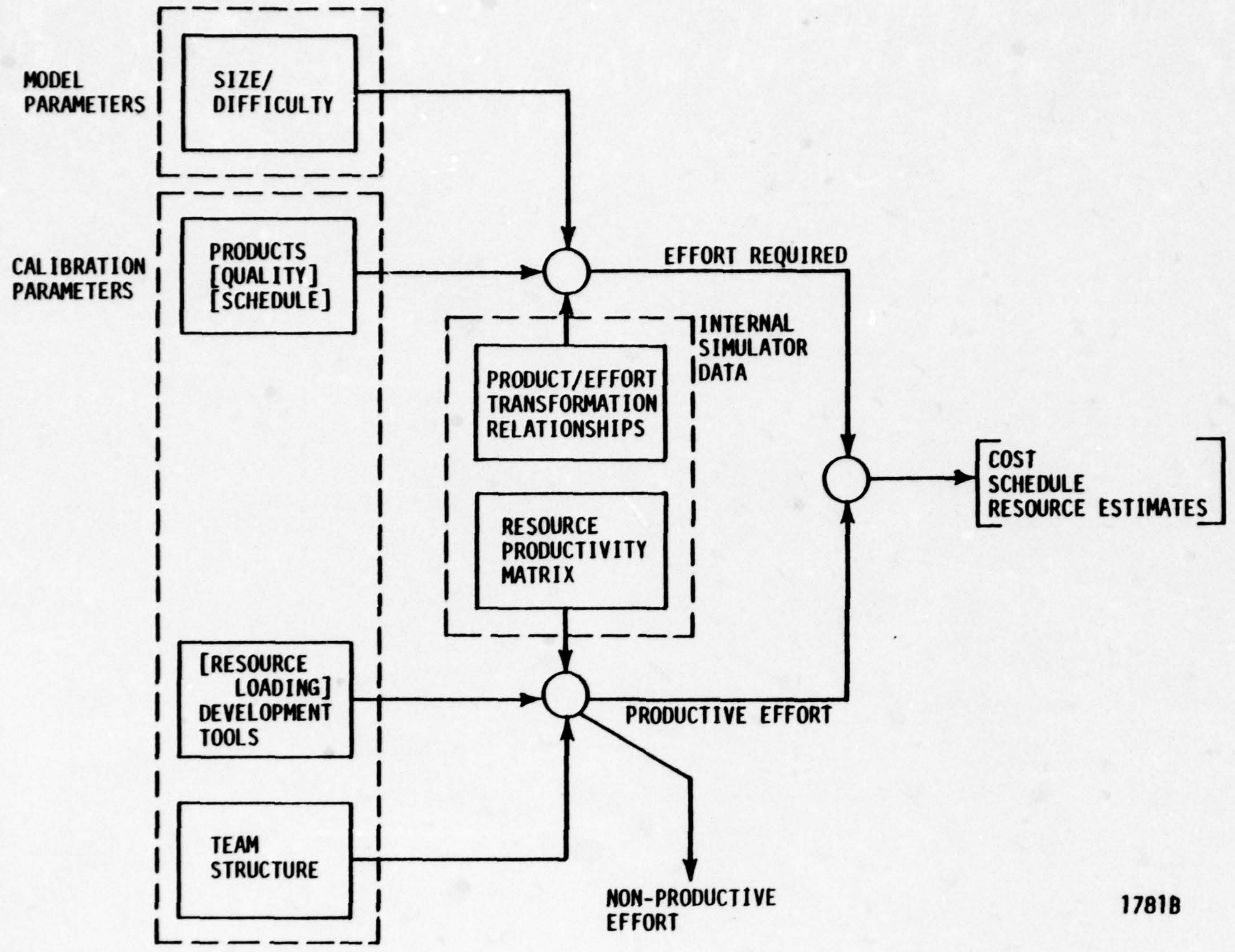
Complexity
- type of application
- processing time requirements
- memory utilization
- hardware dependencies
- rating

<u>RESOURCES</u>

- team composition (type)
- staff mix (personnel type, experience, number)

<u>PRODUCTIVITY</u>

- personnel productivity
- development tool impact
- turnaround time


MODEL PARAMETERS
SIZE/ DIFFICULTY
CALIBRATION PARAMETERS
PRODUCTS [QUALITY] [SCHEDULE]
[RESOURCE LOADING] DEVELOPMENT TOOLS
TEAM STRUCTURE
EFFORT REQUIRED
INTERNAL SIMULATOR DATA
PRODUCT/EFFORT TRANSFORMATION RELATIONSHIPS
RESOURCE PRODUCTIVITY MATRIX
PRODUCTIVE EFFORT
NON-PRODUCTIVE EFFORT
COST SCHEDULE RESOURCE ESTIMATES

1781B

Figure 4-7 Activity Model Concept

taken into account in this portion of the model. The result of the comparison of effort required with productive effort achievable will provide cost and resource loading requirements in the normal mode of operation. As an alternative mode of operation, a resource loading plan can be input and schedule can be output.

The key to the concept is the development of the product/effort transformation relationships and the resource productivity matrix. Our current approach to evolving these is the use of empirical data to derive first approximations and then using the simulator in a calibration mode; i.e., tuning the simulator by comparing simulated results with past historical developments or on-going developments. Currently these activity models are still at a conceptual stage. Very high level models were used in the prototype to be described in the next section.

4.4 MODEL UTILITY

The conceptualization and decomposition of the software development process as a sequence of activities, as shown in Figure 4-6, provides a model which can be used at several levels. At one level, the model can be used as a checklist for planning and progress status. The list of activities typically performed during a software development, and the interaction can be used to plan the activities to be performed in a future development. Once this plan is established, completion of these activities can be used as a status measurement more accurate than the normally imposed milestones.

At the next level, the model can be used as a PERT-COST tool. The current prototype tool that has been developed has the capability with which activity delay times could be modelled as distributions representing worst case, most likely, and best case estimates of the schedules for those activities. The simulation would then result in the calculation of the expected time in which the network of activities would be completed. An enhancement to the PERT-COST approach available with our simulation approach is modelling resource usage as a function of time also.

A third level, that at which the prototype simulator was developed, is a high level process model. At this level, the activities are modelled at a relatively high level. Sensitivities in the development plan and in the assumptions made in the model development could be analyzed. At a high level, impacts of using different techniques and tools could also be analyzed.

The last and most detailed level, is a detailed process model. At this level all of the concepts introduced in paragraph 4.3 would be utilized to model the activities. The analysis capabilities possible in the process model mentioned above would be of greater fidelity due to the finer detail of the activity models. At this level of capability, the full complement of support to the management planning and control of a software development would be provided. This support would span project planning, project control, technology assessment, and contingency planning, as shown in Table 4-5.

Table 4-5
Utility of the Software Development
Process Simulator

- PROJECT PLANNING
  - COST ESTIMATION
  - TIME REQUIREMENTS
  - RESOURCE REQUIREMENTS
- TECHNOLOGY ASSESSMENT
  - ASSESSMENT OF IMPACT OF NEW TOOLS, TECHNIQUES, AND METHODOLOGIES
- PROJECT CONTROL
  - PERFORMANCE ASSESSMENT
  - BOTTLENECK ANALYSIS
  - RESOURCE TRADEOFF ANALYSIS
- CONTINGENCY PLANNING
  - IMPACT ASSESSMENT

SECTION 5

DEVELOPMENT OF A SIMULATOR

5.1 PROTOTYPE DESIGN CONCEPTS

To further evaluate the feasibility of simulating the software development process, a prototype simulator was constructed and demonstrated by modeling a past software development. The prototype was developed with the concept that it could eventually be extended to provide a full software development process simulation capability.

During the construction of the prototype tool, there were several design decisions made. Two of these decisions affecting the implementation of the simulator are described here. These are followed by a general description of the prototype simulator that was built, and the experiment performed using that prototype.

The first decision made was to use a process-oriented view in the simulation instead of the traditional event orientation. An event-oriented view tends to place a simulation at one level of detail, since all state changes in the system being modelled are represented by events that occur at that particular level only. The process-oriented view lends itself to multi-level modelling since process descriptions allow one to describe processes and their relationships as sequences of activities without having to explicitly sequence events, where an activity can easily be a process with its own set of activities.

With a process-oriented view, the system being modelled is viewed as a collection of interacting processes, with the interactions controlled and coordinated by the occurrence of events. The advantages of this view are many. First, a process-oriented model is a more natural way to express the structure of a system. Secondly, the user does not have to define and keep track of the events which signal state changes in the system since the simulator does the event sequencing. Furthermore, a process orientation automatically provides for process structuring. Finally, a process, with all of the event definition and sequencing it implies, can be a subprocess of another process, contain its own subprocess, or be both. This point shows another advantage to the process-view; i.e., that the individual processes that make up a system being modelled can be defined in varying levels of detail, depending on a user's desires and his knowledge of the process in question. Note also that this level of detail does not have to be consistent among the processes in a system for the model to be usable.

The second design decision made for the software development process simulator was to use the technique of path expressions as a simulation tool. Our use of path expressions has closely followed the work of Habermann [HABE 75], who defined a notation for describing the synchronization and coordination among processes to be used as part of a programming language. This work has been expanded and built upon in flow expressions [SHAW 78], used to describe the sequential and concurrent flow of entities through software components, and in event expressions [RIDD 76], used to describe software system behavior. These experiences with path expressions in software system descriptions provided a natural extension to simulation applications.

When simulating large systems, if we assume a process-oriented view of simulation as we have discussed above, one of the greatest problems is being able to explicitly show the interactions among processes in the system, especially at a user level. If one wanted to know the procedural flow through a given set of processes, the only method available was to trace through the actual code of the processes. As path expressions are concerned with the interactions between processes, it was a natural extension to apply them to process-oriented simulation to aid in describing the behavior of systems.

5.2 PROTOTYPE IMPLEMENTATION

With the above design decisions in mind, a prototype software development process simulator was built. A set of simulation tools which we had developed for modelling computer systems, ([WONG 78a], [WONG 78b]), were utilized. Among these were: MORTRAN, a macroprocessor used to provide a FORTRAN preprocessor with structured programming constructs; and, SIMTRAN, a process-oriented simulation language based on GASP IV [PRIT 74]. The relationships between these tools is shown in Figure 5-1.

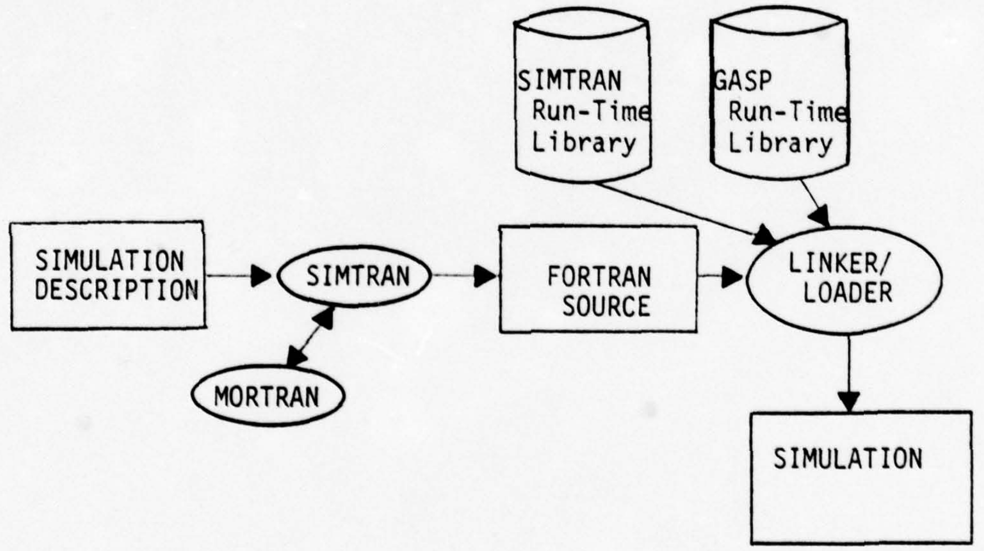


Figure 5-1
Prototype/Tools Relationships

At the core of the prototype is the Path Expression Parser/Interpreter (PEPI). PEPI consists of two basic parts: the Path Expression Interpreter (PEI), and the Path Expression Parser (PEP). The purpose of the Path Expression Parser is to read a user's path expression (written in the Path Expression Language - PEL) and generate a table such that PEI can read the table and control the execution and synchronization of the processes in the simulation. PEI reads the table from PEP, determining which processes are to be started, which order they are to be performed in, and when they are to be started.

Figure 5-2 shows the structure of PEPI. To use the system, a user must provide four things:

1) A PEL source file describing the simulation's path;
2) A list of process names to be used in the model;
3) A SIMTRAN process library for the processes to be used in the simulation;
4) A deck of parameters for a given simulation.

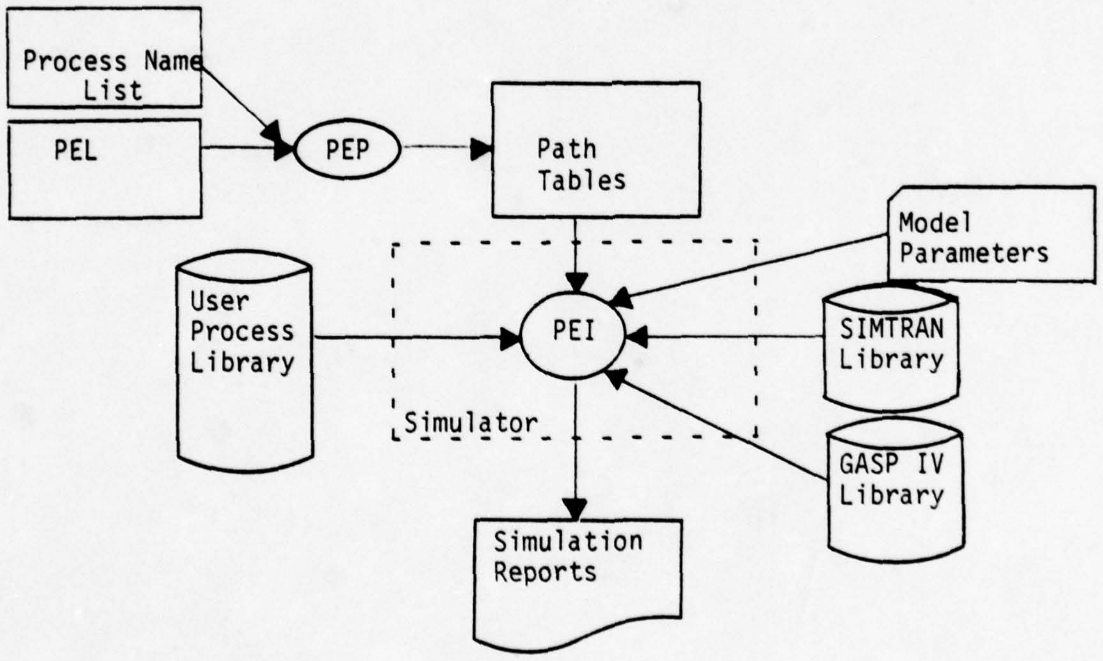


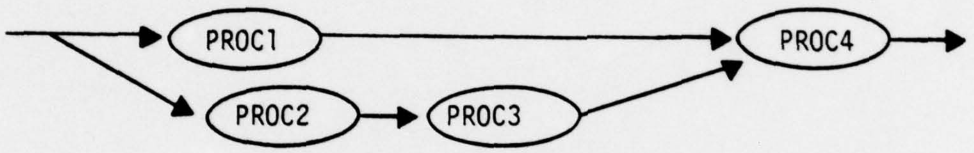
Figure 5-2
Path Expression Parser/Interpreter Overview

Path expressions are a quick and easy way of showing the sequencing of processes, assuming that the processes have been defined. Through a relatively simple syntax, descriptions of parallel and/or sequential processing within the system being described are easily generated.

For example, given:

PROC1:(PROC2,PROC3),PROC4

as a path expression where a comma indicates process sequencing, a colon shows process parallelism, parentheses indicate process groupings, and all strings are process names, a diagram of the same sequence would appear as:



The advantages to this technique are readily apparent. Once a given set of processes has been defined to the level of detail desired, one needs only rearrange the path expression to change the simulation. The implication here is that the generator of the path expression does not have to know about simulation to experiment with a given model. This approach leads to a highly modular simulation, which gives the simulation builder enormous flexibility in adding, deleting, or modifying the processes involved. Further, once a process has been properly defined using this technique, it can be placed in a library of "common" processes. The process can then be used in as many different simulations as applicable by writing the proper path expression. Another advantage to path expressions is inherent in their simplicity: given a set of processes, the two operators":"and",", parentheses, and an understanding of their use, any number of simulations can be easily constructed and, more importantly, readily understood.

PEP will read the process name list and the PEL source to generate the path table for PEI. The process name list is just a list of names, one per card image, where the first six characters on the card are assumed to be process names. There must be a one-to-one relation between the names in the list and the processes in the SIMTRAN process library. PEL has several features of note: 1) it implements the path expressions described before; 2) it is totally free-format; 3) a semi-colon indicates the remainder of a line is comment; and, 4) it has macro-definition capabilities. Notice that macros allow easily repeating processing sequences, and also that macros can contain other macros. Using this syntax, as shown in Figure 5-3, one defines the path expression of the simulation to be performed. This information is then used by PEP to generate path tables. These tables contain information about the sequencing and synchronization of processes that were used in the path expression. PEI, reading the path tables, executes the proper groups of processes as defined in the path expression, so that all sequencing and synchronization is done as defined by the user. However, for PEI to work properly, it first needs the library of SIMTRAN processes. The library must contain a process description for each process name used in the path description. As an option, the user can describe additional processes besides the ones available in the process library. These processes would be described using SIMTRAN [WONG 76]. Finally, GASP will read the parameter deck provided by the user for such control values as the start time, final time, and seeds for random number streams. These parameters are used by GASP and SIMTRAN run-time routines to control the execution of the simulator.

Once a simulation has been performed, PEPI provides certain output reports. The reports from PEP identify any syntax errors in the user's path expression and they display, by a series of tables, the results of the parsing of the path expression. The reports from PEI identify the processes that are initiated. SIMTRAN and GASP IV also provide standard simulation reports. These reports provide the results of the simulation. They contain resource utilization statistics, wait time and queue length statistics, and detailed traces of the simulation. The net result of this information is to tell the user how his model performed under the conditions specified in his parameter deck.

Figure 5-3, below, describes the syntax of the Path Expression Language (PEL) and Figure 5-4 provides an example of the usage of the language to specify a process structure.

Meta Symbols

|	is an alternation of symbols;
[ ]	is zero or more repetitions of the enclosed symbols;
%[]%	indicates logical grouping;
::=	indicates replacement;
-[]-	indicates zero or one occurrences of the enclosed symbols.

PEL Definition

```
<simulation description>::=<heading>-[<macro table>]-<simulation>END
   <heading>::=BEGIN<title>
     <title>::=<character>[<character>]
   <macro table>::=MACRO-TABLE<macro definition>[<macro definition>]
                   END-MACRO
   <macro definition>::=<identifier>=(<path expression>)
      <identifier>::=<character>[<character>]
      <path expression>::=<term>[,<term>]
          <term>::=<factor>[:<factor>]
          <factor>::=<<identifier>>
    <simulation>::= SIMULATE= (<path expression>)
```

Figure 5-3
Backus-Naur-Form for Path Expression Language

```
BEGIN  EXAMPLE 1

     MACRO-TABLE

          A= (X1,Y1)
          B= (X2:X3,Y2,A)
     END-MACRO

     SIMULATE = (A,B:A,X1,X2:Y1)
END   ;EXAMPLE 1
```

SIMULATE statement represents the following network of processes:

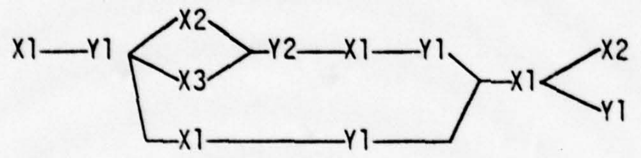


Figure 5-4
Path Expression Language Example

5.3 PURPOSE OF EXPERIMENT

The primary purpose of performing an experiment was to determine the feasibility of the basic capabilities necessary for the simulation of the software development process. In addition, the experiment illustrated the utility of the path expression and process-oriented approaches, and provided some experience during which lessons could be learned and refinements in our approaches could be accomplished.

5.4 DESCRIPTION OF THE EXPERIMENT

The experiment was oriented toward modeling a past large-scale software system development. The simulated results were then compared with the historical data that was maintained about the development effort. This approach to an experiment is more modest than a full validation of the simulation model in which the simulated results would be used to predict the actual results, and a comparison of predicted versus actual would provide a validation criteria. Our experiment was more a calibration of the model to assess if, in fact, a development effort could be modelled to some degree of accuracy. Calibration is utilized by analytic techniques also (RCA PRICE-S and Putnam's SLIM) to tune the analytic model to the development organization. We envision this practice also pertaining to the Software Development Process Simulator, where various parameters or internal tables within the simulator could be tuned to a particular development organization by modeling past developments.

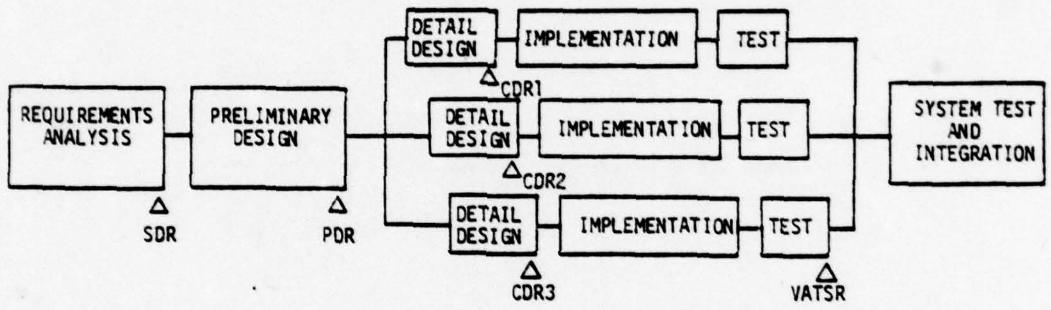
The software system development that was modelled consisted of three major subsystems (or CPCI's). The system was a command and control ground system developed under contract for the Air Force. The three subsystems ranged from 75,000 to 150,000 lines of JOVIAL source code each (including comments). Complete statistics on the development activity were maintained, including the number of design problem reports, software problem reports, and source code statistical profiles, as well as manpower expenditures.

The development was performed in strict accordance with military standards with staggered milestones for each of the subsystems. Most current cost estimation techniques do not model this incremental subsytem development, as shown in Figure 5-5. Our approach to modelling this development technique will be illustrated later in this section. The development that was modelled, however, only consisted of detailed design, implementation, test and integration phases since the CPCI's were modifications of an existing system, and a requirements analysis phase was unnecessary. The flexibility of the simulation approach to model only those phases performed during the development will also be illustrated. Some simplifying assumptions were necessarily made because of the budget and time limitations of the study. No constraints were imposed because of the restrictions of the simulation approach. These limitations to the experiment are described in paragraph 5.5. The details of the experiment model follow.

Four types of resources were modelled. These resources represent the four types of personnel used: System Engineers (SYSE), Analysts (ANLT), Programmers (PROG), and Quality Assurance/Test personnel (QA). Grouped with the quality assurance and test personnel are configuration management personnel, program support librarians, and other support personnel.

The simulation also consisted of two major types of processes, representing different perspectives in the development of a system. The first type views a software system as a whole and models the integration of various CPCI's (Computer Program Configuration Items) into the final deliverable product. Secondly, there are those processes that are performed during the development of a particular subsystem or CPCI, and, hence, view the system as consisting of the particular CPCI. Thus, at a system level, the processes modelled were:

- TSTP: Test Plan Development;
- TPRD: Test Procedure Development;
- SSVT1 & SSVT2: Subsystem Validation and Test.



	LEGEND
SDR	SYSTEM REQUIREMENTS REVIEW
PDR	PRELIMINARY DESIGN REVIEW
CDR	CRITICAL DESIGN REVIEW
VATSR	VALIDATION & ACCEPTANCE TEST SPECIFICATION REVIEW

Figure 5-5 Incremental Subsystem Development

At the subsystem level, or for each CPCI, the processes modelled were:

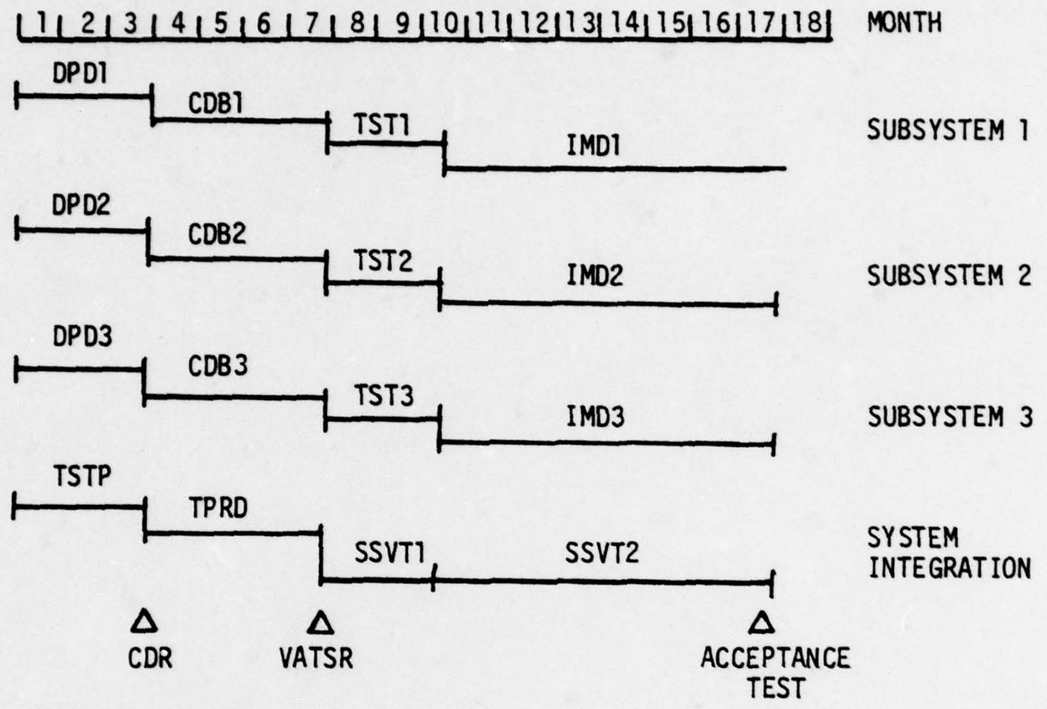
- DPD: Detailed Program Design;
- CDB: Coding and Debug;
- TST: Subsystem Testing;
- IMD: Integration and Maintenance Support.

The motivation for the above breakdown of processes is threefold. First, these processes are the first version of a library of generalized processes that can be used to simulate any software development. To facilitate this approach the processes are parametric, that is, certain variables are passed to the process to represent the specific system being developed. Secondly, we can simulate several CPCI's being developed concurrently with phased milestones, by specifically providing generalized processes for simulating CPCI development. Third, we demonstrate the flexibility allowed with simulation by only modelling the processes performed in the subject development.

For the prototype simulation, there were three individual CPCI's that were all portions of the same system. For convenience, these subsystems were called Subsystem 1 (SS1), Subsystem 2 (SS2), and Subsystem 3 (SS3). Process scheduling and interaction is shown in Figure 5-6 with the corresponding path expression shown in Figure 5-7, as output from PEPI. Note the conciseness with which the information in Figure 5-6 is expressed in Figure 5-7.

5.5 LIMITATIONS OF THE EXPERIMENT

At present, the primary limitation of the experiment is the fact that the level of abstraction of the simulation is high. The relationships internal to the process descriptions representing the activities were not modelled at a very sophisticated level of detail. In the processes that were defined for the prototype model, the approach was to view each activity as a consumer of resources for various amounts of time.



	LEGEND
DPDi	DETAILED PROGRAM DESIGN
CDBi	CODING AND DEBUG
TSTi	SUBSYSTEM TEST
IMDi	INTEGRATION & MAINTENANCE SUPPORT
TSTP	TEST PLAN PREPARATION
TPRD	TEST PROCEDURE DEVELOPMENT
SSVT1 SSVT1	SYSTEM TEST & INTEGRATION
CDR	CRITICAL DESIGN REVIEW
VATSR	VALIDATION & ACCEPTANCE TEST SPECIFICATION REVIEW

Figure 5-6 Process Flow

```
05/29/79                    19.6930                              PAGE      1
                  PATH EXPRESSION PARSER          VERSION 1.5
       BEGIN SAMPLE
       ;
       ;     SDPS SAMPLE MODEL
       ;
       ;         BY    J.A. MCCALL
       ;               A.H. STONE
       ;
       ;         APRIL 1979
       ;
              MACRO-TABLE
                 DESIGN = (DPD1:DPD2:DPD3)
                 CODING = (CDB1:CDB2:CDB3)
                 TEST   = (TST1:TST2:TST3)
                 INTEG  = (IMD1:IMD2:IMD3)
              END-MACRO
       ;
       ;.    BEGIN THE SIMULATION
       ;
              SIMULATE = (DESIGN:TSTP,          ; DESIGN PHASE
                          CODING:TPRD,          ; CODING PHASE
                          TEST:SSVT1,           ; TESTING PHASE
                          INTEG:SSVT2)          ; INTEGRATION PHASE
       ;
       END  ; SAMPLE

            24 LINES PROCESSED
             0 NON-FATAL ERRORS
             0 FATAL ERRORS

--- PROCESSING COMPLETED
```

Figure 5-7 Path Expression

Subsystem size, complexity, project organization, team structures, productivity figures, and turnaround time on the development computer were considered in representing the level of effort required to perform each of the activities. These variables were calibrated to the past development. They were not all controlled by input parameters.

For example, the considerations that were taken into account to model the Coding and Debug (CDB) process for Subsystem 1 (SS1) are listed below:

- Each subsystem or CPCI was developed by a team during this project. The teams were a modification of chief programmer teams and had a hierarchical organization of a system engineer as team leader, one or more analysts, and one or more programmers. This team structure was utilized in the model for the design processes, as well as the coding processes. Eventually we would want to have available other team structures which could be identified parametrically or provide the capability to impose a team structure parametrically. For the prototype, only one team structure was modelled.

- The sizing of the team was accomplished by considering size of the subsystem, complexity of the subsystem, and historical data of turnaround time on the development computer and productivity figures. These variables eventually will be parameters in the formulation of personnel-type requirements. Experience levels for the various personnel types were not taken into account in the prototype, and only the four types of personnel mentioned before were modelled. The sizing calculation for subsystem 1 resulted in a team of a system engineer, two analysts, and two programmers.

- The degree to which the quality of the product resulting from an activity affected subsequent phases was only modelled at a gross level utilizing the problem report rate to size the maintenance effort level.

The processes were parametric, in that resource levels and timing variables were parameters, and used as library functions in the simulation; however, further extensions are required before they could be used generally.

The input and output capabilities of the prototype simulator were very limited. A minimum number of inputs were required; most variables were incorporated in the process descriptions. The output capabilities were restricted to those provided by PEPI, SIMTRAN, and GASP IV. No additional output reports were generated. The outputs are illustrated in the next section.

5.6 RESULTS OF THE EXPERIMENT

This experiment was performed in a calibration mode. There were records available from a past development effort performed at General Electric for the Air Force, as was described in Section 5.4, and the simulation was based on the historical data available. The manner in which this calibration was performed was to use this data from the development project to drive the simulation in an effort to produce results that approximated other data from the project. At a summary level, the results of our simulation are shown in Figure 5-8. The line labelled "ACTUAL" is the graph of the data from the actual Air Force project superimposed on the graphs of the simulation results. Cursory examination of the data shows a clear correspondence between observed and experimental values.

Comparison of the graphs in Figure 5-8 shows that there is a 4.98% error between the areas underneath the observed and experimental data curves. These results are considered quite acceptable. Some of the peaks of spikes seen in the actual data can be attributed to five-week fiscal months, which plotted at a granularity of one month causes higher manpower expenditures to be illustrated.

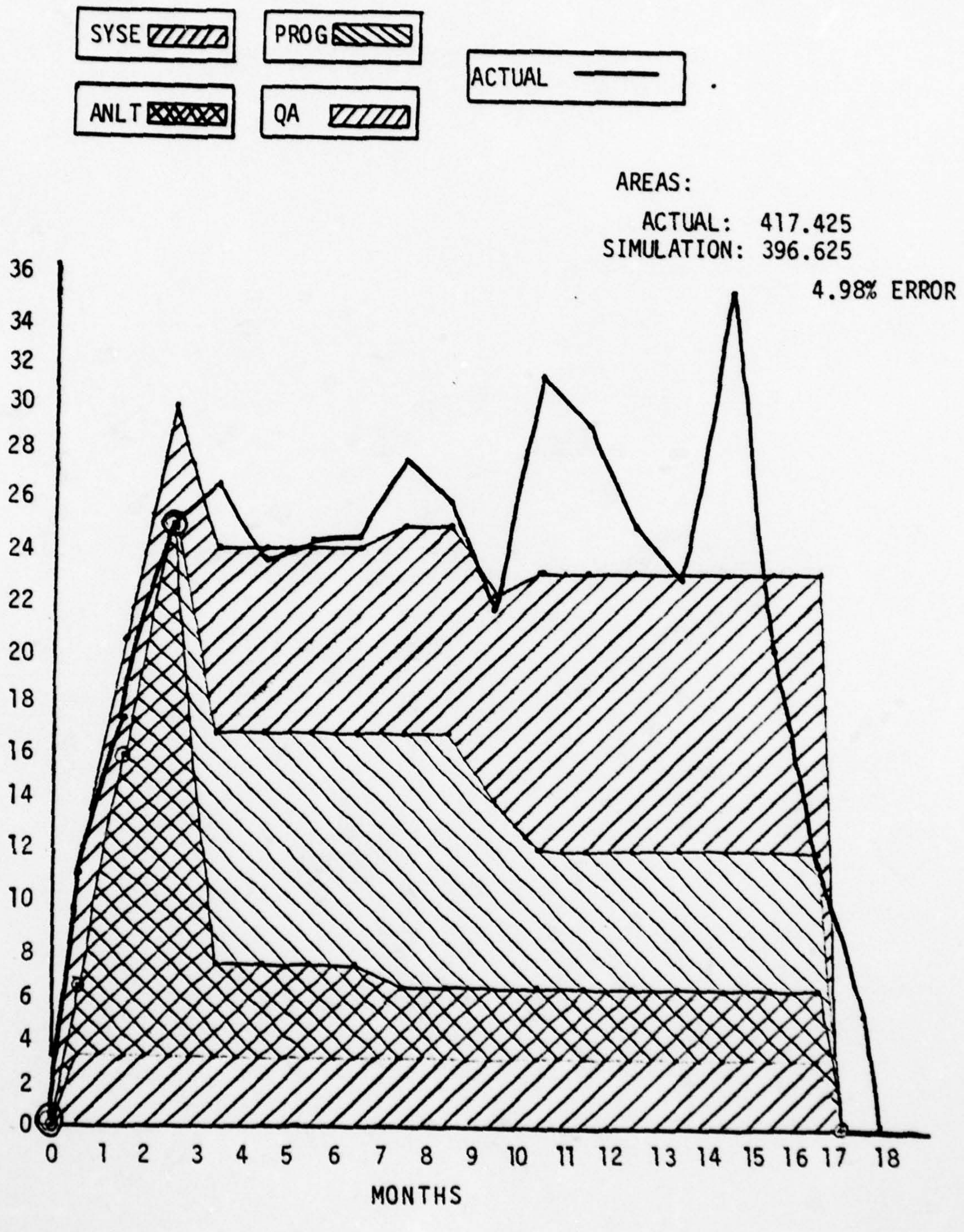


Figure 5-8 Experiment Results

In addition to these plots, resource utilization and queue statistics are provided and also a dynamic trace of the complete simulation. These data are available for performing detailed analyses of bottlenecks in the development and impacts for changes.

SECTION 6

CONCLUSIONS

6.1 SUMMARY OF SIGNIFICANT RESULTS

Our research has covered the spectrum from concept formation to analysis to experimentation. First, we addressed the problem of how to apply simulation techniques to study the software development process. Then, we investigated what the characteristics of software developments are, based on the "world-view" established by our simulation approach. Finally, we studied when our modelling methodology is valid by learning how to design simulation experiments based on our modelling approach. Table 6-1 summarizes our major accomplishments, and indicates where we go from here.

Table 6-1

Conclusions Matrix

	ACCOMPLISHED	FUTURE
(1)	SIMULATION APPROACH	
	Combined activity-product model forms conceptual basis.	Identification of simulation variables, model rules, model inputs and outputs.
(2)	PROCESS DECOMPOSITION	
	Activity-product network demonstrates practical application.	Detailed specification of activities, products, factors, and resources.
(3)	SIMULATOR DEVELOPMENT	
	Simulator prototype demonstrates experimental feasibility.	Data collection to support full experiment.

6.2 EVALUATION OF EXPERIMENT

The experiment provided not only a demonstration of some of the modelling concepts and approaches established during this study but also, provided the basis for evaluation of our ideas. The accuracy of the results, substantiating the ability to calibrate the model, were very promising. The use of a process-oriented simulation language and the path expression interface to the simulator provided a very easy-to-use and flexible simulation tool. This is important from two viewpoints. First, the tool, to be effective, must be planner/analyst-oriented rather than simulation-oriented. The path expression language provides an automated way to layout a PERT-like plan of the network of activities to be performed. The underlying process-oriented simulator, driven by the path expression, then provides modelling capabilities far more powerful than the probability distributions for schedules provided in a PERT system.

The ease of selecting and interconnecting the activities is a key concept because it facilitates tailoring the model to the specific development. Analytic techniques, in comparison, are developed based on past developments without differentiating between projects. For example, there is no differentiation between a project that uses simulation during design and one that does not, or between a project that has a formal requirements analysis phase, one that does not. The analytic techniques tend to be a gross approximation of the software development. The activity models allow tailoring of the model to the actual development. Only those activities to be performed during the development are selected.

The addition of activity models which are not in the library of generic activity models is easy. In the prototype simulator this was accomplished by writing a process using SIMTRAN and including its reference in the path expression. Inclusion of acquisition office activities could be accomplished in this manner.

The framework provided by the model allows experimentation/evaluation of the individual activity models. The framework should also aid the estimation process.

The simulation run time and cost was insignificant. The simulation tools developed and utilized have evolved over a number of years, and are oriented specifically to modelling computer system applications.

The concept of activity/product progression models of the activities shows promise in accounting for quality and documentation considerations in the software development process. However, they were not modelled in the experiment in enough detail, or for all of the activities to provide a thorough evaluation.

A true test of the model would be to use it as a prediction tool and then as a status assessment tool during an actual development. Once in the development, the model could be used to evaluate progress by substituting actual data at the end of a phase into the model and conduct a simulation. The impact on the rest of the project plan, based on the actuals at that time, would be provided by the simulator.

Based on the experiment results and the modelling concepts that have evolved, we conclude that it is feasible to model the software development process. In addition, the insight gained from the model can have significant benefits to the management of the development effort. These benefits will be derived from a better understanding of the software development process through research and evaluation using the model of development techniques, organizations, and methodologies. And, the benefits will be derived from the use of the model as an automated tool supporting cost estimation (planning), and project progress assessment (control). In order to realize those benefits, some additional research tasks must be conducted. These tasks are identified in the next paragraph.

6.3 TOPICS FOR FURTHER INVESTIGATION

The topics for further investigation fall into three areas: (1) development of the simulator; (2) experimentation and demonstration of the simulator; and, (3) further basic research.

The prototype simulator developed during this study was constructed to demonstrate our approach to modelling the software development process, and to assist in the assessment of the feasibility of the approach. In addition, extensibility was a major design goal of the prototype; i.e., we attempted to develop the prototype so that we could expand it into a full-capability simulator. The development of the Path Expression Parser/Interpreter, based on the SIMTRAN simulation language, provides the core of the future simulator. The expansion required includes development of the library of activity models and enhancement of the input/output capabilities. The activity models used in the prototype were not modelled at a very detailed level or were not as parametric as required for general use. Also, not all activities identified in Section 3 were modelled. The input/output capabilities of the prototype relied on the plotting and histogram output capabilities of GASP IV, the standard simulation reports of SIMTRAN, and the developed path expression parser for input. The path expression capabilities need to be expanded to allow more network expressions to be modelled, and management-oriented reports need to be generated.

Once the simulator has been developed, we feel it should be calibrated in an Air Force Systems Programs Office environment. This calibration would involve modelling past developments and turning the model to the results of those developments. After calibration, the simulator should be utilized during an actual development in an experimental model; i.e., during the use of the simulator, evaluation of its utility should be conducted.

Further basic research is required also. Prior to the development of the full complement of the library of activity models, further research into modelling some of the activities at a greater level of detail is needed.

This includes, not only developing the models, but supporting that development with data collection during actual software development projects.

SECTION 7
BIBLIOGRAPHY

The Bibliography contains references listed alphabetically according to four categories:

- Software Project Management (Models);
- Simulation and Modelling;
- Cost Estimation; and,
- Others.

SOFTWARE PROJECT MANAGEMENT (MODELS)

ANAI 76 "An Air Force Guide to Software Documentation Requirements", W.L. Schoeffel, MITRE Corporation, June 1976.

ARON 74 Aron, J.D., The Program Development Process, Addison-Wesley Publishing Company, Menlo Park, California, 1974.

BELA 76 Belady, L.A., et al., "A Model of Large Program Development", IBM Systems Journal, Vol. 15, No. 3, 1976.

BOSC 78 Bosch, J.A., Briggs, P., "Software Development for Fly-by-Wire Flight Control Systems", GE TIS, 1978.

CARR 75 Carrow, J., et al., "Workbook on Structured Programming", NTIS, February 1975.

CAUD 77 Caudili, R., "Understanding the Development Life Cycle", 1977 National Computer Conference.

COSG 78 Cosgrove, D., "ESD Software Acquisition Process Model Concept and Feasibility", MITRE Working Paper-21981, November 1978.

ELYE 77 Ely, E.H., "Software Management: A Dynamic Approach", Defense Systems Management College, May 1977.

GEHR 76 Gehring, P., "A Quantitative Analysis of Estimating Accuracy in Software Development", NTIS AD A-047 674, August 1976.

GORD 78 Gordon, S.C., "The Development of a Computer Software Management Discipline", Proceedings of the 1978 National Aerospace and Electronics Conference, 1978.

HAGA 75	Hagan, S.R., et al. "An Air Force Guide for Monitoring and Reporting Software Development Status", MITRE Corporation, September 1975.
HIEM 75	Hieman, P., Programming Methodology -- Lecture Notes in Computer Science, Springer -- Verlang, New York, 1975.
KOLA 76	Kolasheski, R.F., "An Investigation into the Feasibility of Using the Leontief Input -- Output Model in the Quantitative Management of Computer Programming", USASCS-AT-76-08, 1976.
LIFE 74	"Life Cycle System Management Models for Army Systems", Draft DA Pamphlet No. 11-25, April 1974.
LOVE 76	Love, Tom, "A Review of the Variables Which Influence the Software Development Process", GE TIS 761SP001, 28 June 1976.
LOVE 77	Love, T., Fitzsimmons, A., "A Survey of Software Practitioners to Identify Critical Factors in the Software Development Process", GE TIS 771SP002, 25 January 1977.
METZ 73	Metzger, P., Managing A Programming Project, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
PODO 77	Podolsky, J.L., "Horace Builds a Cycle", Datamation, Nov. 1977.
SACK 70	Sackman, H., Man-Computer Problem Solving, Auerback Publishers, Inc., Princeton, New Jersey, 1970.
SCOT 75	Scott, R., Simmons, D., "Predicting Programming Group Productivity - A Communications Model", First Annual Software Engineering Conference, 1975.
SOFT 77	"Software Phenomenology", U.S. Army Institute for Research in Management Information and Computer Science, Working Papers of the Software Life Cycle Management Workshop, Airlie House, Virginia, 21-23 August 1977.
TURN 76	Turn, R., Davis, M., Reinstedt, R., "A Management Approach to the Development of Computer-Based Systems", International Conference on Software Engineering, October 1976.
WILL 76	Willworth, N.E., "Software Data Collection Study: Survey of Project Managers", RADC-TR-76-329, December 1976.

SIMULATION AND MODELLING

COOK 68	Cook, W.H., "Decision Analysis for Product Development", IEEE Transactions on Systems Science and Cybernetics, Vol. SSC-4, No. 3, September 1968.
GREE 67	Greene, James H., Operations Planning and Control, Richard D. Irwin, Inc., 1967.
HABE 75	Habermann, A.N., "Path Expressions", Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, June 1975.
MAHM 77	Mahmoud, M.S., "Multilevel Systems Control and Applications: A Survey", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-7, No. 3, March 1977.
MALC 62	Malcolm, D., "System Simulation -- A Fundamental Tool for Industrial Engineering", Simulation in Social Science, Prentice-Hall, Inc., Englewood Cliffs, New York, 1962.
MCCA 76	McCall, J., Seyfarth, T., Wong, G., "SIMTRAN User's Guide", Software Engineering Laboratory Standard No. 46, October 1976.
MCCA 78	McCall, J., "Information and Data System Simulator", 1978 Summer Computer Simulation Conference, July 1978.
PRIT 74	Pritsker, A.A.B., The GASP IV Simulation Language, John Wiley & Sons, Inc., New York, 1974.
RIDD 76	Riddle, W.E., "An Approach to Software System Modelling, Behavior Specification and Analysis", RSSM/25, Dept. of Computer and Communication Sciences, Univ. of Michigan, July 1976.
SHAW 78	Shaw, A.C., "Software Descriptions With Flow Expressions", IEEE Transactions of Software Engineering, Vol. SE-4, No. 3, May 1978.
SMAL 68	Smallwood, R.D., "A Decision Analysis Of Model Selection", IEEE Transactions on Systems Science and Cybernetics, Vol. SSC-4, No. 3, September 1968.
WILL 74	Willis, R., "Structured Model Development Techniques", Symposium on the Simulation of Computer Systems, NBS, June 1974.
WONG 75	Wong, G., McCall, J., Seyfarth, T., "Computer Network Simulation", GE TIS 75CIS07, December 1975.

WONG 78a Wong, G.Y., "Computer System Simulation With GASP IV", 78CIS009, General Electric, June 1978.

WONG 78b Wong, G.Y., "Design Methodology for Computer System Modelling Tools", paper presented at Symposium on Modelling and Simulation Methodology, Rehovet, Israel, August 1978.

COST ESTIMATION

ARON 69 Aron, J.D., Estimating Resources for Large Programming Systems, IBM, 1969.

ASOF 77 "A Software Resource Macroestimating Procedure", HQ, Department of the Army, DA Pamphlet No. 18-8, February 1977.

AUTO 78 "Automatic Data Processing Resource Estimating Procedures (ADREP)", Planning Research Corporation, PRC R-1527, August 1970.

BOUR 78 Bourdon, G.A., Duquette, J.A., "A Computerized Model for Estimating Software Life Cycle Costs", ESD-TR-77-253, April 1978.

CLAP 76 Clapp, J., "A Review of Cost Estimation Methods", MITRE Corporation, ESD-TR-76-271, August 1976.

COST 72a "Cost Analysis: Program Breakdown Structure and Codes", AFSCM 173-4, Department of Air Force, November 1972.

COST 72b "Cost Estimating Procedures", Department of the Air Force, HQ, Air Force System Command, AFSCM 173-1, 17 April 1972.

DEVE 76 Devenny, T.J., "An Exploratory Study of Software Cost Estimating at the Electronics Systems Division", NTIS, AD-A1030-162, July 1976.

DOTY 77 Doty, D.L., Nelson, P.J., Stewart, K.R., "Software Cost Estimation Study Guidelines for Improved Software Cost Estimation", RADC-TR-77-220, Vol. I and Vol. II, August 1977.

FARR 64 Farr, Leonard, et al., "Cost Aspects of Computer Programming for Command and Control", System Development Corporation, NTIS AD 430259, 13 January 1964.

FIND 74 Findley, R.A., "Computer Software Development Costs, Predictable or Not", NTIS AS-A-039 730, May 1974.

FINF 78 Finfer, M., Mish R., "Software Acquisition Management Guidebook: Cost Estimation and Measurement", ESD-TR-78-140, March 1978.

FLEI 66 Fleishman, T., "Current Results from the Analysis of Cost Data for Computer Programming", System Development Corporation, AD 637801, August 1966.

GEHR 76 Gehring, Jr., Lt. Col. Philip F., USAF, "Improving Software Development Estimates of Time and Cost", Second International Conference on Software Engineering, San Francisco, 13 October 1976.

GOVE 74 "Government/Industry Software Sizing and Costing Workshop -- Summary Notes", USAFESD, 1-2 October 1974.

GRAV 76 Graver, C.A., et al., "Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software", GRC, ESD TR, November 1976.

HANS 76 Hansen, D.L., "Software CER Feasibility Study", HQ, SAMSO, Cost Analysis Division, December 1976.

JUNK 78 Junk, W., McCall, J., Putnam, L., Walters, G., "Survey of Software Cost Estimation Techniques", GE TIS 78CIS010, May 1978.

LABO 66 LaBolle, V., "Development of Equations of Estimating the Costs of Computer Program Production", System Development Corporation, AD 637 760, June 1966.

MANA 71 "Management Information Systems: Handbook of ADP Resource Estimating Techniques", US Army, TB 18 19-3, August 1971.

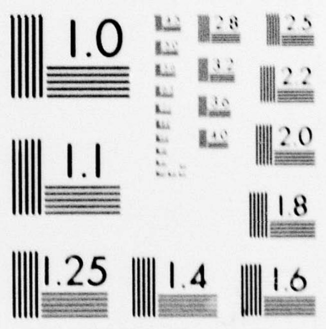
MORI 74 Morin, L.H., "Estimation of Resources for Computer Programming Projects", University of North Carolina, 1974.

NELS 66 Nelson, E.A., "Methods of Obtaining Estimates of Computer Programming Costs: A Taxonomy", System Development Corporation, AD 665 478, August 1966.

NELS 67a Nelson, E.A., "Management Handbook for the Estimation of Computer Programming Costs", System Development Corporation, AD 648 750, 20 March 1967.

NELS 67b Nelson, E.A., et al., "Cost Reporting for Development of Information Processing Systems", System Development Corporation, April 1967.

NORD 77 Norden, Peter V., "Project Life Cycle Modeling: Background and Application of the Life Cycle Curves", Software Life Cycle Management Workshop, Airlie House, 21-23 August 1977.


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

ORTH 77 Orth P.J., "Development of On-Line Software Package for Calculating Acquisition Costs", NTIS AD-A052 714, April 1977.

PIET 70 Pietrasanta, A.M., Resource Analysis of Computer Program System Development on the Management of Computer Programming, Auerback Publishers, Princeton, New Jersey, 1970.

PUTN 77a Putnam, Lawrence H., "The Influence of the Time-Difficulty Factor in Large Scale Software Development", IEEE COMPCON 1977, 9 September 1977.

PUTN 77b Putnam, Lawrence H., Wolverton, Ray W., "Quantitative Management: Software Cost Estimating", IEEE Computer Society Tutorial, Computer Software and Application Conference, 8-11 November 1977.

PUTN 78a Putnam, Lawrence H., "Information Required to Support Sizing, Estimating and Control of Software Life Cycle", COMPCON 1978, 2 March 1978 and AIIE, 3 April 1978.

PUTN 78b Putnam, L.H., "Measurement Data to Support Sizing Estimating and Control of the Software Life Cycle", GE internal paper, Jan. 1978.

REFE 77 "Reference Manual - PRICE Software Model", RCA PRICE Systems, Cherry Hill, New Jersey, December 1977.

SCHN 77 Schneider, J., "A Preliminary Calibration of the RCA PRICE-S Software Cost Estimation Model", NTIS AD-A046 808, September 1977.

SCHN 78 Schneider, V., "Prediction of Software Effort and Project Duration -- Four New Formulas", ACM SIGPLAN Notices, Vol. 13, No. 6, June 1978.

SMIT 75 Smith, R., "Estimating Software Project Resource Requirements", (Structured Programming Series, IBM), RADC-TR-74-300, Vol. XI, January 1975.

STEP 76 Stephenson, W.E., " An Analysis of the Resources Used in the SAFEGUARD System Software Development", International Conference on Software Engineering, October 1976.

WALS 77 Walston, C.E., Felix, C.P., "A Method of Programming Measurement and Estimation", IBM Systems Journal, No. 1, 1977.

WEIN 66 Weinwurm, G.F., "Data Elements for a Cost Reporting System for Computer Program Development", System Development Corporation, AD 637 804, August 1966.

WOLV 74 Wolverton, Ray W., "The Cost of Developing Large-Scale Software", IEEE Transactions on Computers, Vol. 23, No. 6, 1974.

OTHERS

AIRF 66 "Air Force ADP Experience Handbook", Planning Research Corporation, December 1966.

BLAC 77 Black, Katz, Gray, Curnow, "BCS Software Production Data", RADC-TR-77-116, March 1977.

BOLE 76 Bolen, N., "Air Force Guide to Contracting for Software Acquisition", AD-A020 444, January 1976.

BROO 75 Brooks, F.P., The Mythical Man-Month, Addison-Wesley Publishing Company, Reading, Mass., 1975.

CARR 75 Carrow, J., Reaser, J., "Interactive Programming: Summary of an Evaluation and Some Management Considerations", USACSC-AT-74003, March 1975.

DODD 76 "DOD Defense System Software Management Program", OASD, March 1976.

DONE 77 Donelson, "Project Planning and Control", Datamation, 1977.

EMBE 77 "Embedded Computer Resources and the DSARC Process - A Guidebook", OSD, 1977.

FIND 75 "Findings and Recommendations of the Joint Logistics Commanders", Software Reliability Working Group, November 1975.

FINF 76 Finfer, M., "Software Data Collection Study: Data Requirements for Productivity and Reliability Studies", RADC-TR-76-329, December 1976.

JONE 78 Jones, T.C., "Measuring Programming Quality and Productivity", IBM Systems Journal, Vol. 17, No. 1, 1978.

KATZ 76 Katzan, H., Systems Design and Documentation, Van Nostrand Reinhold Co., New York 1976.

KIRK 73 Kirk, F.G., Total System Development for Information Systems, John Wiley & Sons, New York, 1973.

KOSY 74	Kosy, D., "Air Force Command and Control Information Processing in the 1980's: Trends in Software Technology", Rand, 1974.
LIAS 74	Lias, E.J., "On-Line Versus Batch Costs", Datamation, Dec. 1974.
LIEN 75	Lientz, B.P., Swansco, E.B., Tomplins, G.E., "Characteristics of Application Software Maintenance", Graduate School of Management, University of CA at Los Angeles, partially sponsored under ONR N00014-75-C-0266, 1975.
MANA 76	"Management of Computer Resources in Major Defense Systems", Department of Defense Directive No. 5000.29, April 1976.
MCCA 77	McCall, J., Richards, P., Walters, G., "Factors in Software Quality", RADC TR-77-369, November 1977.
MYER 78	Myers, Ware, "The Need for Software Engineering", Computer, IEEE Computer Society, Vol. II, No. 2, February 1978.
NATO 69	"NATO Science Committee Report", January 1969.
NAVA 76	NAVAIRINST 5230.5, "Responsibility and Requirements for Preparation of Software Life-Cycle Management Plans (SLCMP)", 21 July 1976.
NELS 77	Nelson, R., "RADC Data Repository", NASA/GODDARD Software Engineering Workshop, September 1977.
PARI 76	Pariseau, R.J., "Improved Software Productivity for Military Computer Systems through Structured Programming", NADC-76044-50, March 1976.
PROC 73	Proceedings of a Symposium on the High Cost of Software, Sept. 1973.
PROC 75	Proceedings of the International Conference on Reliable Software, 1975.
SCOT 74	Scott, R., Simmons, D., "Programmer Productivity and the Delphi Techniques", Datamation, May 1974.
SOFT 75	"Software Acquisition Management Guidebook: Regulations, Specifications, and Standards", MITRE Corporation, October 1975.
THAY 75	Thayer, T.A., "Understanding Software Through Empirical Reliability Analysis", Proceedings of the 1975 National Computer Conference, 1975.

WALT 78 Walters, G., McCall, J., "The Development of Metrics for Software R&M", Proceedings of Annual Reliability and Maintainability Symposium, January 1978.